UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

SENG365 Web Computing Architecture: #4 Aspects of HTTP servers, REST and GraphQL

Ben Adams
Course Coordinator & Lecturer
benjamin.adams@canterbury.ac.nz
310, Erskine Building

The story so far…

In the lectures

- Introduction to web computing
  - e.g. client-server
- HTTP, REST, & Intro to APIs
- Some JavaScript concepts
- Introduction to the assignments
- Asynchronous behaviours

In the labs

- Self-study labs x 3
- Introduction to Node.js
- Introduction to persistence
- Structuring your server application

In the lecture this week

- News and admin
- More REST
- HTTP idempotent and safe methods: side effects
- API versioning
- RESTful GraphQL
- CAP and ACID
- GraphQL

Labs

Term 1 (6 weeks)

- 3 x pre-labs (self-study)
- Week 2: Lab 1
- Week 3: Lab 2
- **Week 4: Lab 3**
- Week 5 & 6: assignment support

Term 2 (6 weeks)

- Week 7: Lab 4
- Week 8: Lab 5
- Week 9: Lab 6
- Week 10 & 11: assignment support
- Week 12: **compulsory** lab
  - Assignment 2 testing

Assessment

The assessment

- Assignment 1 (25%)
  - **No extension**
- Mid-semester test (20%)
  - **Wed 24 March, 7pm**
  - **E8 Lecture Theatre**
- Assignment 2 (25%)
  - Compulsory lab in final week
  - **No extension**
- Exam (30%; 2hrs)

Additional information and requirements

- Assignment resources on Learn
- API specification with skeleton project
- Infrastructure
  - eng-git project
  - MariaDB (MySQL) server
  - Docker VM server
  - ‘Behavioural server’

Exam

- Topics:
  - HTTP (requests, responses)
  - REST APIs
  - Javascript (code and modules)
  - Web databases
- Format:
  - Paper exam, closed book, no notes
  - 24 short answer questions
  - 2 hours (likely will not take you the full time)

A REST service is…

- Platform-independent
  - Server is Unix, client is a Mac…
- Language-independent
  - C# can talk to Java, etc.
- Standards-based (runs *on top of* HTTP*), and
- Can easily be used in the presence of firewalls

- (*REST ≠ HTTP)

- RESTful systems typically, but not always:
- … communicate over the Hypertext Transfer Protocol (HTTP)…
- … with the same HTTP verbs (`GET`, `POST`, `PUT`, `DELETE`, etc.) used by web browsers…
- … to retrieve web pages and send data to remote servers.

- RESTful apps are (a subset of) web apps

(HTTP | REST) resources

- Nouns not verbs (because resources are things not actions)
- Instances or Collections

- Resource instance typically identified by :id
  - Integer
  - UUID
  - See also Hypertext As The Engine Of Application State

CRUD and REST (continued)

- For a partial change, use PATCH (like diff)
- Include only elements that are changing
- Null (value) to delete an element
- See also “best practices”
  - http://51elliot.blogspot.co.nz/2014/05/rest-api-best-practices-3-partial.html and
- JSON merge patch
  - https://tools.ietf.org/html/rfc7386

How good are each of these three examples?

- You're going to add a new user.
- How good / appropriate are the following three approaches for adding a user?

1

```
GET /adduser?name=Robert HTTP/1.1
```

2

```
POST /users HTTP/1.1
Host: myserver
Content-Type:
application/xml
<?xml version="1.0"?>
<user>
  <name>Robert</name>
</user>
```

3

```
POST /users HTTP/1.1
Host: myserver
Content-Type:
application/json
{
      "name": "Robert"
}
```

REST features

- REST offers no built-in security features, encryption, session management, QoS guarantees, etc.
- … these can be added by building on top of HTTP
- For encryption, REST can be used on top of HTTPS (secure sockets)
  - For security, see later lectures

REST vs SOAP

- **REST** has displaced **SOAP**, because…
- … REST is considerably easier to use
  - e.g. SOAP has a ‘heavy’ infrastructure
- … works nicely with AJAX / XHR
  - e.g. XML is verbose; JSON is more concise
- … has some network advantages
  - accepted through firewalls

Stateless requests

- A complete, independent request doesn't require the server, while processing the request, to retrieve any kind of application context or state.
- A RESTful Web service application (or client) includes within the HTTP headers and body of a request all of the parameters, context, and data needed by the server-side component to generate a response.
- The entire resource is returned, not a part of it.
  - Design resources carefully
- Statelessness:
  - Improves Web service performance
  - Simplifies the design and implementation of server-side components…
  - because the absence of state on the server removes the need to synchronize session data with an external application.

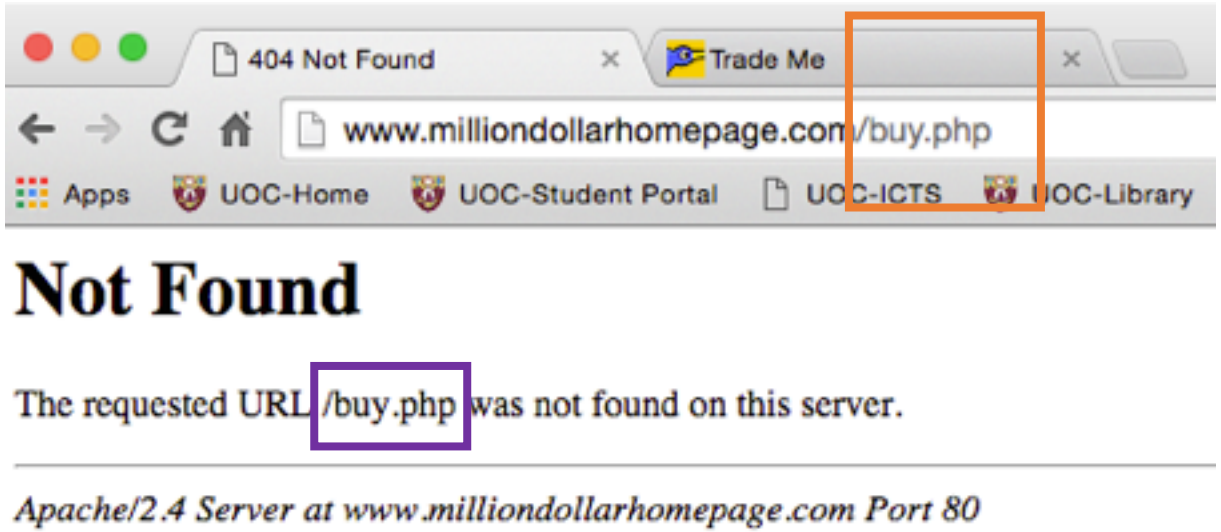
RESTful URLs

- **Hide the server-side scripting technology** file extensions (.jsp, .php, .asp), if any, so you can port to something else without changing the URIs
  - in fact, **hide all implementation details!**
- **Be consistent** in the singularity / plurality of resource names
  - Basically use **singular** or **plural**, but don't mix them
  - `/student` & `/course`; not `/student` and `/courses`
- Keep everything **lowercase**
- Substitute **spaces for hyphens**
- Instead of using the 404/Not Found code if the request URI is for a partial path, always provide a default page or resource as a response
  - Though we don't expect this in the assignment…

From an earlier example

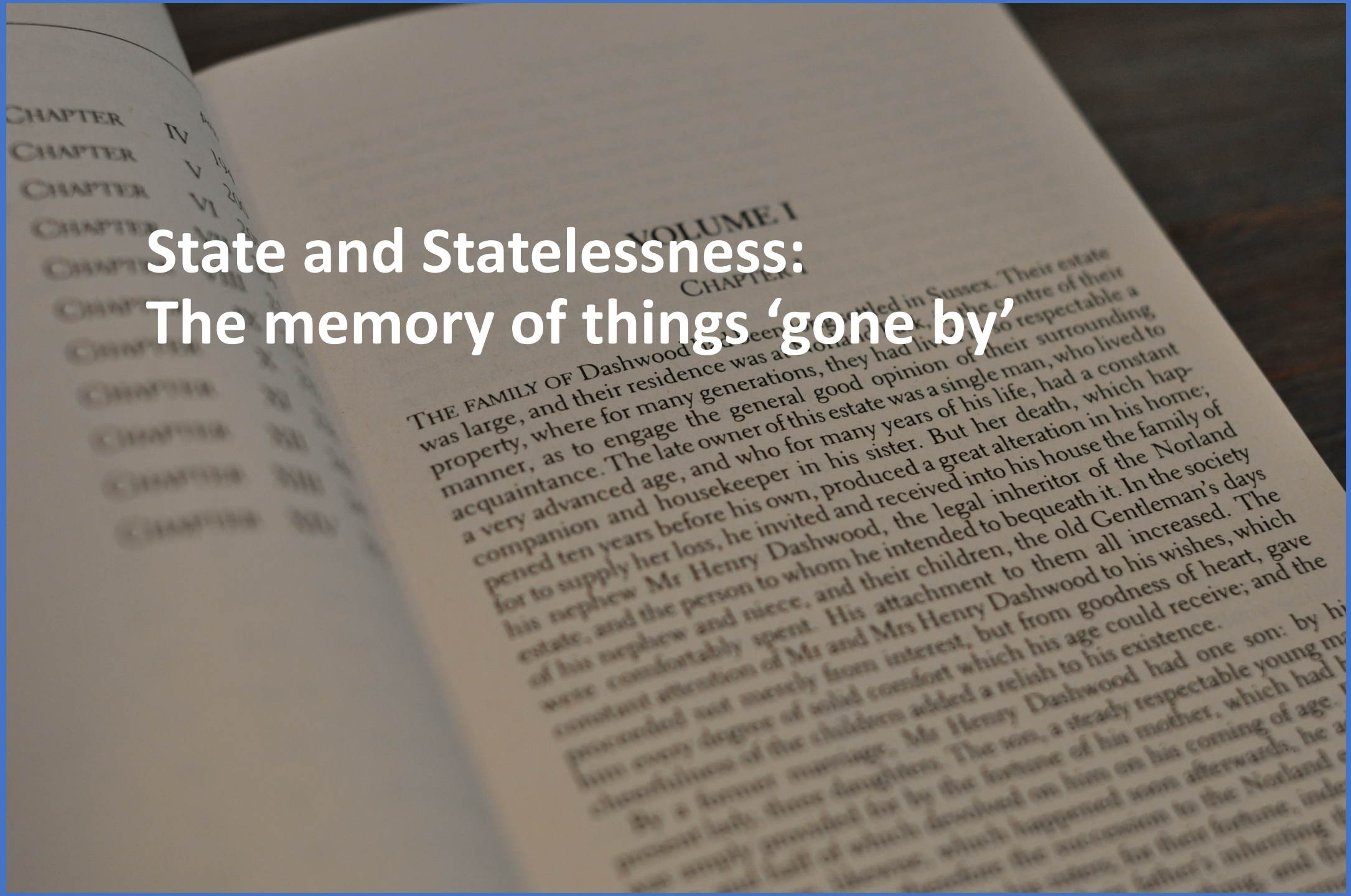
The Million Dollar homepage

Implementation detail in the API



REST issues

- Tightly **coupled to HTTP**
- Request-response
  - Multiple request-responses needed
  - **Underfetching** and **overfetching**
  - Latency increases for *full set* of request-response
    - (GraphQL to the rescue…?)
  - Implied tree-structure



State and Statelessness: The memory of things 'gone by'

https://www.flickr.com/photos/wastes/6119262807/in/photolist-ajJPDa

Mini-overview

- State timescales
- Session (state) information
- GET parameters e.g. after the ?
  - HTTP bodies & cookies
    - Types
    - Sequence
    - Limitations
    - Session IDs
- (ACID)

State timescales

- Individual HTTP request (stateless)
- Business transaction
- Session
- Preferences
- Record state

shortest

longest

Session (state) information

- For *web applications* (in contrast to public websites or webpages) there is a need to maintain some stateful information about the client.

GET ? parameters

- Maintain some kind of session variable in the parameter to the HTTP request
- Variable does not contain the username and password, but a unique (‘random’) identifier
- Include variable as a parameter in **each** network request e.g.,

```
GET www.example.com?sessionid=<var>
```

- Why is this ‘bad practice’?
- Why may something like this be needed, at times?

Cookie

- Use cookies to maintain session information
- The server issues a unique (‘random’) identifier in the cookie to the client, for that username & password
- Client sends back the cookie with **each** network request to the server
  - e.g. in the POST data
- Note that the username and password are not sent (once the user is logged on).

Cookies

- A **small piece of data** initially sent by the server to the client.
  - Comprises name-value pairs
  - Also has attributes (that are not sent back to the server)
    - Expiry ‘date’ (duration)
- Used to **maintain state information**
  - e.g. items in a shopping basket
    (although this example may be better kept on the server)
  - e.g. browser activity such as a ‘path’ through a registration process

Types of cookie

First-party cookie	A cookie set by the server to which the browser primarily connects.
Session cookie	Exists only for the duration of that browser session, and the browser typically deletes the cookie
Persistent cookie (aka tracking cookie)	Persistent data. The cookie is not deleted when the browser closes. Can be used by advertising to track user behaviour. Can be used to store credentials e.g. log in details.
Secure cookie	A cookie that can only be transmitted over an encrypted connection, such as HTTPS.
HTTPOnly cookie	Can only be transmitted through HTTP/S, and are not accessible through non-HTTP APIs such as JavaScript.
Third-party cookie	Cookies set by third-parties that serve content to the page e.g. advertising.

Sequence of cookie-ing

- Request from browser
- `GET /index.htm HTTP/1.1`
- `Host: www.example.com`

- Response from server
- `HTTP/1.1 200 OK`
- `Content-type: text/html`
- `Set-cookie: sessionToken=a1b2c3; Expires = [dat]`

- Follow-up request from browser
- `GET /profile.htm HTTP/1.1`
- `Host: www.example.com`
- `Cookie: sessionToken=a1b2c3`

Those cookies (review)

- Use cookies to maintain information
- Send the cookie with **each** network request
  - e.g. in the POST data

- What's the security risk with including the cookie in the POST data?
- How might we address this risk?

Limitations (?) of cookies

- Each browser maintains its own ‘cookie jar’
- A cookie does not identify a person
- A cookie identifies the combination of:
  - User account
  - Web browser
  - Device
- A cookie requires that the browser is cookie-enabled and is set to allow cookies

HTTP

HTTP…

- **HTTP is a stateless protocol** originally designed for document retrieval
  - HTTP requests and responses are self-contained
  - Not dependent on previous requests and responses
- **HTTP common commands** are:
  - **GET** – retrieve a named resource (shouldn't alter visible server state)
  - **HEAD** – like GET but only gets headers
  - **POST** – submits data to the server, usually from an HTML form
- Some **less common commands**, but sometimes used in RESTful web services
  - **OPTIONS** – get supported methods for given resource
  - **POST** – create a new resource
  - **PUT** – submit (changes to) a specified resource
  - **DELETE** – delete a specified resource
  - **PATCH** – modify an existing resource

Idem-what? *Idempotence*, and *safe* methods

- Clarification
  - This is about best practice & standards
  - You can break these standards, but shouldn't
  - Concerns the *server*, not the client
- *Safe* methods are HTTP methods that do not modify resources (not a static resource)
  - Like read-only methods
- *Idempotent* methods: the *intended* state (e.g. data changed) is the same whether the method is called once or many times
  - Depends on the state of the system
  - Depends on what you're actually doing

HTTP method	Should be safe	Should be Idempotent
GET	Yes	Yes
HEAD	Yes	Yes
PUT (GET first)	No	Yes
POST	No	No
DELETE	No	Yes
PATCH	No	No

https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

Why does idempotency matter? Side-effects

“Idempotency and safety (nullipotentcy) are ***guarantees*** that **server applications make to their clients** … An operation doesn’t **automatically become idempotent** or safe just because it is invoked using the GET method, if it isn’t **implement**ed in an idempotent manner.

A poorly written server application might use GET methods to update a record in the database or to send a message to a friend … **This is a really, really bad design.**

Adhering to the idempotency and safety contract helps make an API fault-tolerant and robust. ”

https://codeahoy.com/2016/06/30/idempotent-and-safe-http-methods-why-do-they-matter/

CAP and ACID

CAP theorem*

applies to distributed systems that store state

https://www.flickr.com/photos/bk1bennett/6334043117

Definitions

Consistency - A read is guaranteed to return the most recent write for a given client.

Availability - A non-failing node will return a reasonable response within a reasonable amount of time (no error or timeout).

Partition Tolerance - The system will continue to function when distributed system partitions (fragments).

The principle is:

- You can have any two of the three {C,A,P}

But the reality is:

- Networks fail.
- Your web app has **no control** on network failure.
- You have to tolerate partitioning (P)
  - Asynchronous behavior…

*Eric Brewer, Principles of Distributed Systems 2000

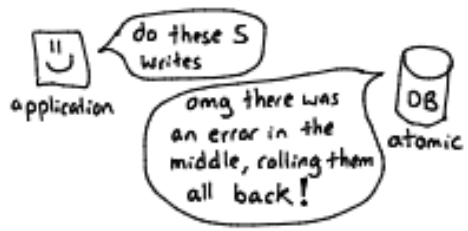
What's ACID?

notes from Martin Kleppmann's *amazing*
"Designing Data-Intensive Applications" book

ACID is about safety guarantees for database transactions.

Atomicity

not about concurrent writes
(that's "isolation")



Consistency

super overloaded term.
This sense of "consistency" is actually an application property not a DB property.

not linearizability
not as in "eventual consistency"

About preserving application invaciants like "every sale gets an invoice".

Isolation

Isolation is about preventing race conditions like this.

Some isolation levels:

- serializability
- snapshot isolation
- read committed

Durability

Perfect durability doesn't exist.

Can involve:

- write-ahead log (usually)
- replication

https://drawings.jvns.ca/drawings/acid.svg

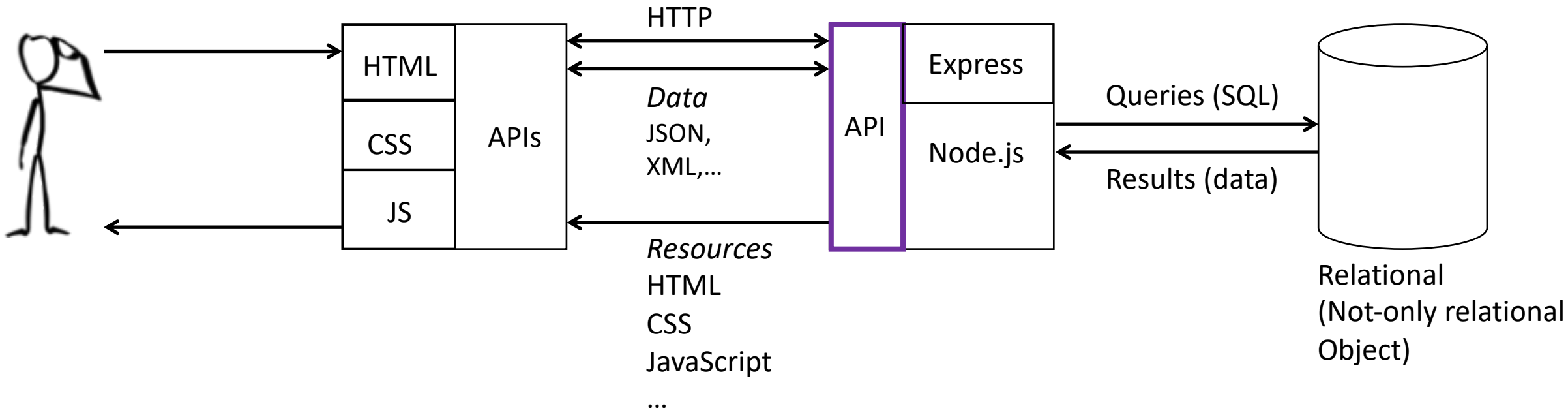
ACID, CAP: not the same

ACID	**CAP**
Atomicity	Yes, a Good Thing
Consistency (preserve invariants)	**C**onsistency (looks like a single-copy)
Isolation	**P**artition+Availability?
Durability	**A**vailability+Consistency?

UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

API Versioning

Reference model for SENG365: `api/v1/`



User	Client	Server	Database
Human	Machine	Machine	Machine

API versioning: mini-overview

- Compatibility between API provider and API consumer
- Semantic versioning
- Specifying API versions in HTTP requests (and handling of those)
- JSON
- Publishing APIs

Interesting (though a bit dated) discussion at SO:

https://stackoverflow.com/questions/389169/best-practices-for-api-versioning

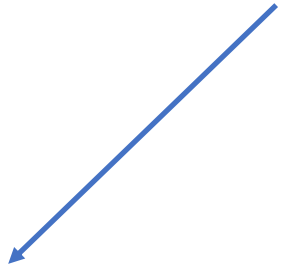
Why version?

- Maintain compatibility of APIs with API consumers, as APIs change
- Add / amend functionality/services
- Performance improvements e.g., reduce the number of HTTP requests
- Market testing of ideas e.g., A/B testing
- System testing e.g. dev, test, prod,
- Subsets of clients, e.g.,
  - B2B vs B2C
  - Geographical / political region
- Rollback of (unacceptable) API

- From client perspective:
  - API is backward compatible if client can continue through service changes
  - Forward compatible if client can be changed without needing service change

Semantic versioning: `MAJOR.MINOR.PATCH`

1. Caching - want different versions cached differently
2. Number or name?
   - If number, what format? Semver? Counter?



Semantic Versioning – semver.org

Given a version number MAJOR.MINOR.PATCH, increment the:

- MAJOR version when you make **incompatible** API changes,
- MINOR version when you add functionality in a backwards-compatible manner, and
- PATCH version when you make backwards-compatible bug fixes.

"Facebook's Marketing API now supports both versioning and migrations so that app builders can roll out changes over time. Read on to understand how you are affected by versions, how to use those versions in our Marketing APIs and Ads SDKs, and what migration windows are.

While Facebook's Platform has a core and extended versioning model, starting Oct 30th, 2014, Facebook's Marketing API will move to a versioning scheme to manage changes in the Marketing API. With Marketing API versioning, all breaking changes will be rolled up into a new version. Multiple versions of Marketing APIs or Ads SDKs can exist at the same time with different functionality in each version."

https://developers.facebook.com/docs/marketing-api/versions

Specifying API version in HTTP request

1. Query parameter
   - `?v=xx.xx`, or `?version=xx.xx` or `?Version=2015-10-01`
   - e.g., Amazon, NetFlix
2. URI
   - `api/v1/`
   - e.g. Facebook: https://graph.facebook.com/v2.2/me/adaccounts
   - Semantically messy (implies version refers to version of object)
3. Header
   - Accept header - hard to test - can't just click on link or type URL
   - Custom request header - duplicates Accept header function
   - https://developer.github.com/v3/media/

Further information e.g., https://blog.pivotal.io/labs/labs/api-versioning

Publishing your API

Credibility

“If a developer doesn’t believe you, then good luck getting them to use your product.”

Support

“Developers are looking for signs of support. Something at some point is going to go wrong, so developers want to know they can solve their problems quickly.”

Success

“Look to share details about how your API can help them achieve something. A lot of companies don’t have that understanding that providing an API is going to create a win for both of you.”

http://www.programmableweb.com/news/how-to-maximize-developer-adoption/analysis/2014/11/17

Publishing your API

- **Documentation** - current, accurate, easy, guide/tutorial/directed (management tool generated)
- **Direct access** (no SDK required)
  - e.g., through Postman or curl (say, `curl -L http://127.0.0.1:4001/v2/keys/message-XPUT -d value="Hello world")`
- SDKs/Samples in **developer preferred languages**
  - Any SDK is just libraries to access REST/SOAP API, nothing more. Potentially an impediment to simply making use of the straight API.
  - Straightforward install and use
- **Free/Freemium** use for developers
- **Instant API keys**
- **Simple sandbox** to try things out for developers

- Before API available, establish **API landing page** on web to discover interest and potential user types

{REST}

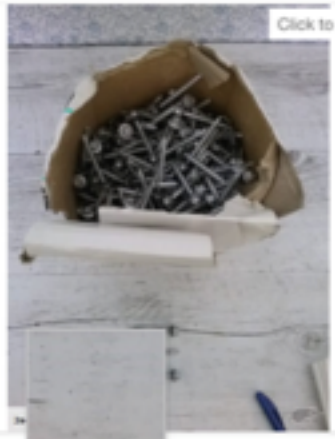
GraphQL

6cm Screws

Listing #: 1570425811
Hastings, Hawke's Bay, NZ

40 people added this to their Watchlist

Pay Now



Closes: 1 min
Tue 20 Mar, 3:34 pm
This auction may auto-extend.

Add to Watchlist

Email reminders: None

Current bid $14.04
Total Bids: 18 Reserve met

Next minimum bid $14.54

Auto-bid Place bid

Hide bid history

Tue 20 Mar

$14.04	baxter04 (1,708)	3:32 pm
$13.54	baxter04 (1,708)	3:32 pm
$13.04	stonepen... (348)	3:32 pm
$12.54	baxter04 (1,708)	3:22 pm
$12.04	rhyco1 (257)	2:47 pm
$11.54	ikipike (198)	9:04 am
$11.04	goodbusi... (244)	7:12 am

Mon 19 Mar

$10.54	ikipike (198)	8:26 pm
$10.50	jimmyjam... (298)	8:26 pm
$6.00	jimmyjam... (298)	8:25 pm

Only the latest 10 bids are shown

Shipping
From $6.50 More...

Buyer Protection Learn More

Description | Questions & answers 2

Maybe around 300 screws

RULES OF ENGAGEMENT---

No warranty implied or given.
All items are sold "as is" and the $1 reserve reflects this.
*Please let us know if you would like more photos on any of our listings.

Pick up: Whakatu, Hastings (Mon-Fri 8.00am-4.30pm)
Items must be collected within 7 days after the Auction closes. Items that have not been collected within two weeks will be relisted unless special arrangements have been made.

Postage: Due to the high volume of sales, we will post winning items out on Wednesdays and Fridays. Once payment is received we will send the items on the next scheduled postage date.

We add items daily so be sure to save us as a seller "3scavengers" and while you're here, check out our other listings.

*Please read the **questions and answers** for this listing.*

Shipping options	
Nationwide	$6.50
Rural	$10.20

Seller allows pick-ups
Seller location: **Hastings, Hawke's**

Payment options

Use your debit or credit card.
What's Pay Now?

14 Classic Fire Engines and memorabilia collection

Listing #: 1571234651
Wellington City, Wellington, NZ

825 people added this to their Watchlist

Closes: Wed 28 Mar, 12:00 pm
(7 days, 20 hours, 30 minutes)

Add to Watchlist

Email reminders: 12 hrs

Start price $150,000.00
No reserve

Starting bid $150,000.00

Auto-bid Place bid

Shipping
Buyer must pick-up More...

Key details

On road costs Excluded
Additional costs may apply

Description | Questions & answers 25

Have you ever wanted to be a fire fighter or even own your own fire engine? Well I did and after 35 years of collecting I have decided to sell my personal collection and find a new passion in life!

I purchased my first fire engine back in 2000 and started a collection now known as the "Wellington Fire Museum" I have many items including fire helmets and uniforms, waterway equipment, Ladders. As there is too much to list individually I would recommend viewing. Please also see the photographs to give you an idea of what is available.

Featured on Stuff, paste link below to find out more
https://www.stuff.co.nz/national/101973760/passion-loses-its-spark-as-fire-engine-collector-puts-his-museum-up-for-sale

For more details check out the Wellington Fire Museum Facebook page

Fire engines as picture (2 -15)

One Business Mobile:
3 months money back guarantee
One month term
$15 per user, per month on selected Red+ Business plans
Find out more >
T&Cs apply
vodafone

Specialist Cars buyer's checklist
- Check the history and ownership with MotorWeb
- How much to insure this classic vehicle?
- How much to get this delivered?
- Check out the Motorsport NZ regulations

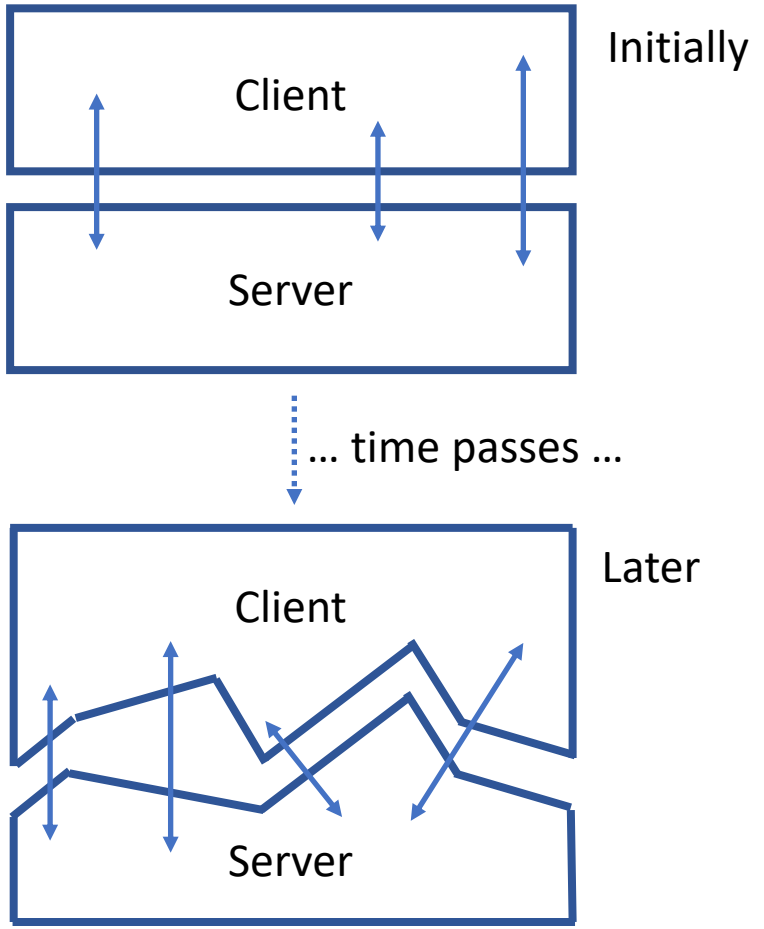
ecostore
safer for you and our world

Endpoints and client views

- Endpoints tend to be designed and structured according to the views expected to be needed on the front-end
  - e.g. we design request parameters (query & body) and the response's JSON structure to fit the view
- That's an efficient design…

… EXCEPT THAT…

- Views change
- Users want different information, more information, less information, more and less views
- The fit between endpoint/s and view/s therefore disintegrates



RESTful APIs and their limitations

1. Fetching complicated data structures requires multiple round trips between the client and server.
2. For mobile applications operating in variable network conditions, these multiple roundtrips are highly undesirable.

An example set of requests

```
/auctions/{id}
/auctions/{id}/bids
/users/…
/auctions/{id}/photos
```

Overfetching and underfetching

Overfetch: Download *more* data than you need

- e.g. you might only need a list of usernames, but /users downloads
- (as a JSON object) more data than just usernames
- And endpoint provides more than you need

Underfetch: download less than you need so must then do more (the n+1 problem)

- e.g. you need a list of most recent three friends for a username, so for *each* item in /users you need to get information from /user/friends, but then only take the first three entries

RESTful APIs and their limitations

3. REST endpoints are usually weakly-typed and lack machine-readable metadata.

(JavaScript is weakly typed too…)

An example of the confusion

`auctionStarttime integer`

Why integer and not Date?

Mapping from integer to date and time?

`POST /auctions` API, is `startingBid` the same as the `auction_startingprice` in the `auction` table?

GraphQL

- A *specification* for:
  - How you specify data
    (cf. strong-typing)
  - How you query that data

- There are reference *implementations* of the GraphQL specification
  - https://github.com/graphql/graphql-js (Node.js)

- Extra lab on LEARN (not pre-req for assignment)

A *very* simple example

Comments

- `Character` is a GraphQL Object Type that has fields
- `name` and `appearsIn` are the fields
- `String` is a scalar type (a base type that's irreducible)
- `[Episode]!` is an array [] that's non-nullable (due to the !)
- Each `type Query` specifies an entry point for every GraphQL query.

Example (of API)

```
type Character {
  name: String!
  appearsIn: [Episode]!
}

type Query {
      hero: Character
}
```

GraphQL vs REST

GraphQL

- Define objects and fields that can be query-able
- Define *entry points* for a query
- The client application can dynamically ‘compose’ the content of the query
- A much more flexible interface to the server side.

REST

- *Endpoints* that are set and inflexible
- Pre-defined fixed endpoints that
  - Require pre-defined inputs
  - Return pre-defined data structures
- Those endpoints are then ‘set’…
  - … until version x.y.z of the API

GraphQL vs REST response code and errors

GraphQL

- All GraphQL queries return 200 response code, even errors.
  - E.g. malformed query, query does not match schema, etc.
- Errors are returned in user-defined field
- Network errors can still return 4xx/5xx
  - E.g. GraphQL server is down

REST

- HTTP response code indicates success / error
- 2xx, 4xx, 5xx, etc.

GraphQL vs REST response code and errors

GraphQL

- All GraphQL queries return 200 response code, even errors.

```
{
  "data": {
    "getInt": 12,
    "getString": null
  },
  "errors": [
    {
      "message": "Failed to get string!",
      // ...additional fields...
    }
  ]
}
```

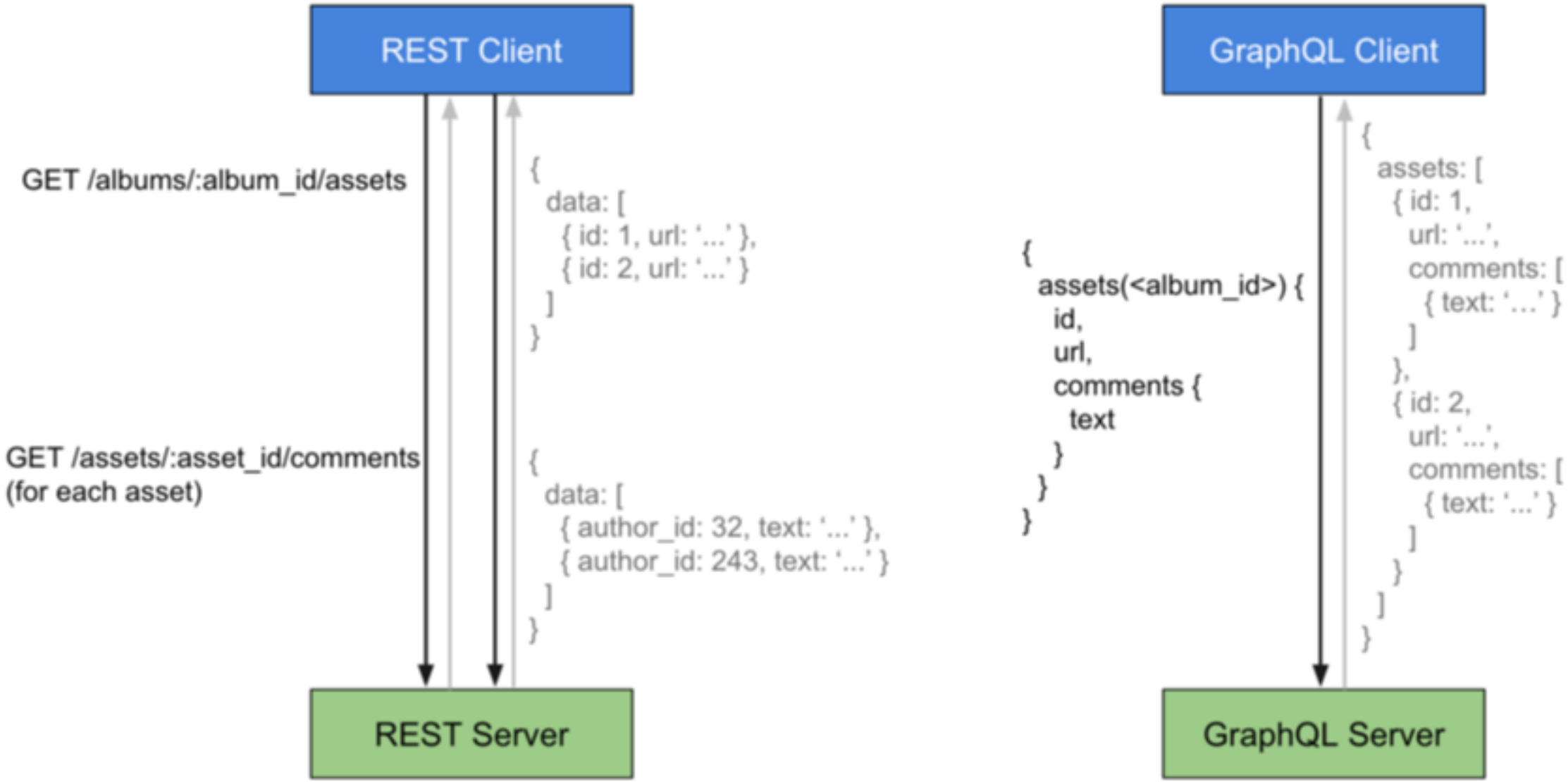
REST

- HTTP response code indicates success / error

== 400 BAD REQUEST

GraphQL

- Does not require you to think in terms of graphs
  - (Though relational tables are a type of graph…)
  - You think in terms of JSON-like structures for a query (see earlier slide)
- Is not querying the database directly
  - Rather is a ‘language’ (specification) for composing queries to a server
- Still requires some kind of pre-defined data and queries on the server-side
  - Objects, fields and allowable queries
  - But these pre-definitions are more ‘atomic’ in their nature



https://medium.com/chute-engineering/graphql-in-the-age-of-rest-apis-b10f2bf09bba

GraphQL uses GETs and POSTs

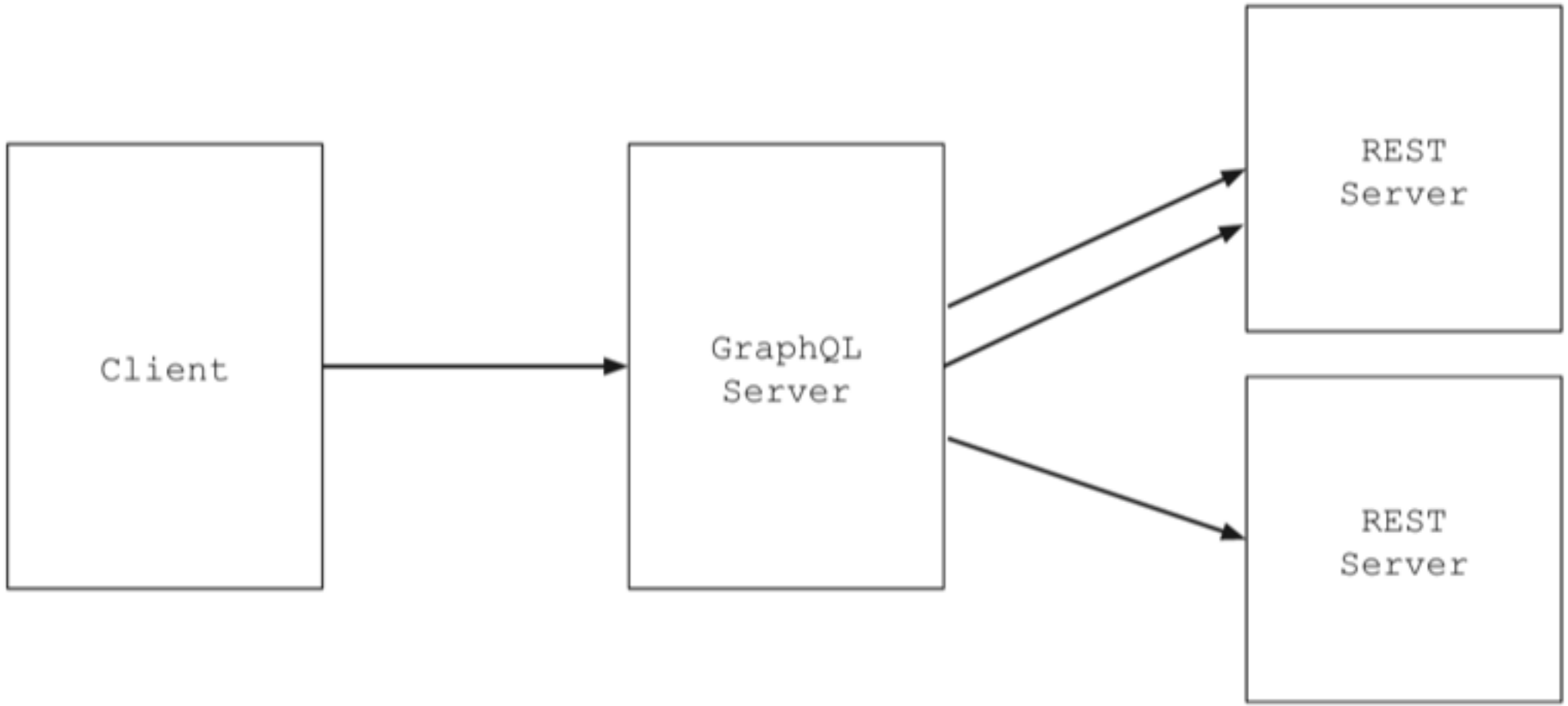
For GETs

- `http://myapi/graphql?query={me{name}}`
- GraphQL query is specified using the URL query parameters (JSON templating…)

For POSTs:

- `http://myapi/query`
- Specify the query in the HTTP body, using JSON, e.g.

```
"query": "...",
"operationName": "...",
"variables": {
"myVariable":
"someValue", ... } }
```



https://medium.com/chute-engineering/graphql-in-the-age-of-rest-apis-b10f2bf09bba

https://www.npmjs.com/package/express-graphql

```
const express = require('express');
const graphqlHTTP = require('express-graphql');

const app = express();

app.use('/graphql', graphqlHTTP({
  schema: MyGraphQLSchema,
  graphiql: true
}));

app.listen(4000);
```

Further resources

GraphQL Introduction

- https://graphql.org

Apollo GraphQL Server

- https://www.apollographql.com/docs/apollo-server/

From REST to GraphQL

- https://0x2a.sh/from-rest-to-graphql-b4e95e94c26b

GraphQL in the age of REST APIs

- https://medium.com/chute-engineering/graphql-in-the-age-of-rest-apis-b10f2bf09bba

UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

SENG365 Web Computing Architecture: #4 Aspects of HTTP servers, REST and GraphQL

Ben Adams
Course Coordinator & Lecturer
benjamin.adams@canterbury.ac.nz
310, Erskine Building