6.1040: Software Design

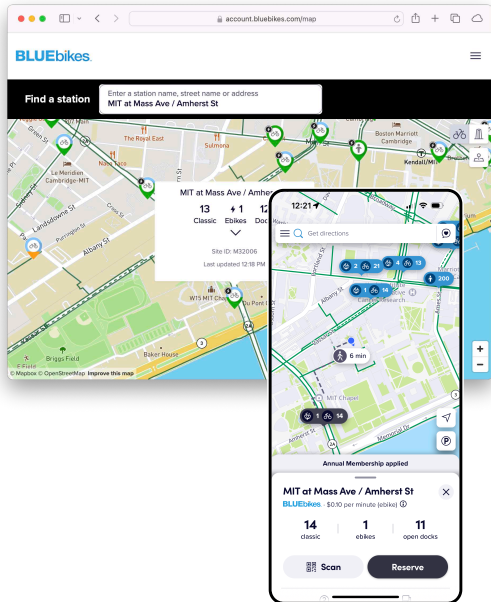
Service Design

Arvind Satyanarayan & Max Goldman

with material by Daniel Jackson

Fall ’24

Client / Server

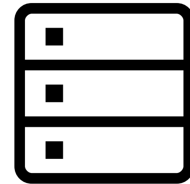


HTTP request

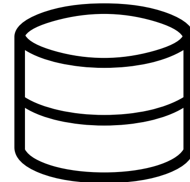
This interface requires design

› URLs › requests › responses

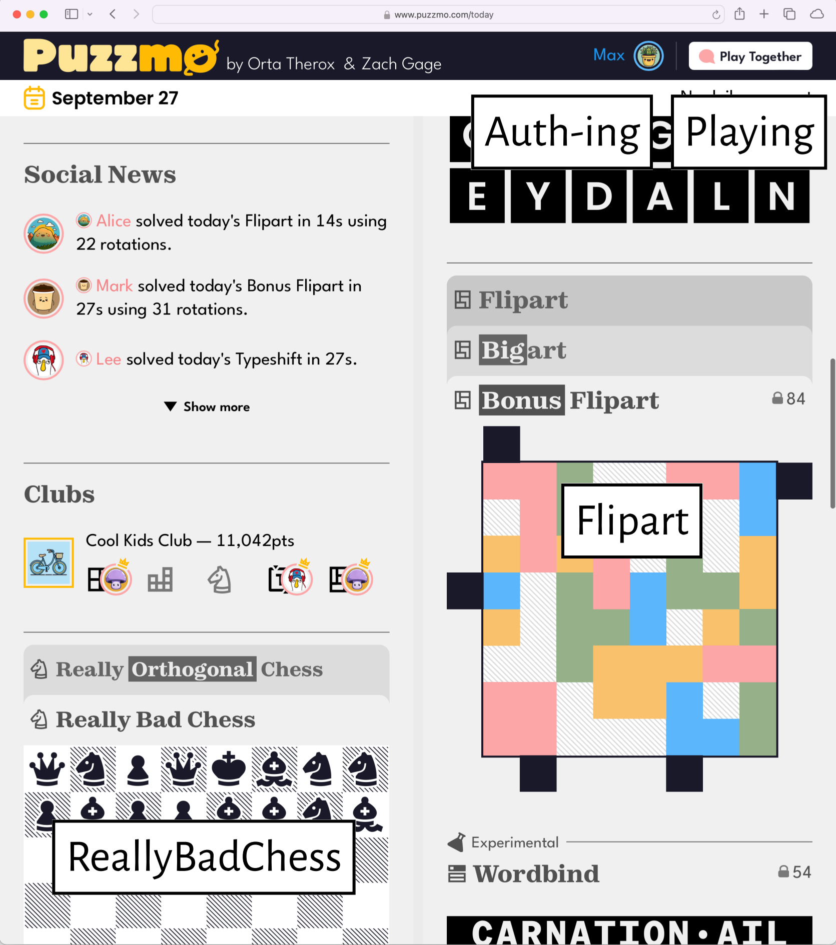
HTTP response



process request & build response

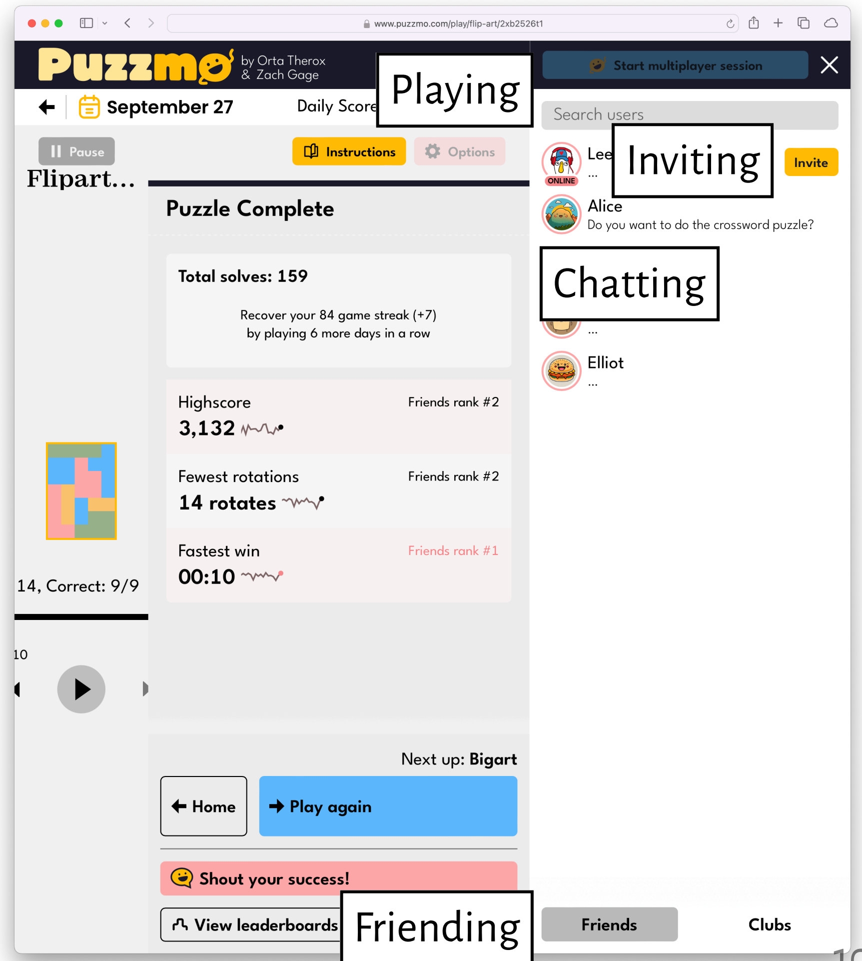


Pitch: social puzzle gaming
with leaderboards (for friendly competition)
and collaborative play (for friendly cooperation)



Pitch: social puzzle gaming
with leaderboards (for friendly competition)
and collaborative play (for friendly cooperation)

A question: is *inviting* a concept,
or an action in another concept?



Puzzmo

concept Friending [User]
purpose ...
principle ...

state

friends: User → **set** User

actions

friend (a: User, b: User)

a.friends += b ; b.friends += a
-or-
add (a, b) and (b, a) to friends
-or-
...

other actions...

concept Inviting [User]
purpose ...
principle ...

state

invites: **set** Invitation
from: invites → User
to: invites → User

actions

invite (s: User, r: User, **out** inv: Invitation)

invites += inv ; inv.from := s ; inv.to := r

accept (r: User, inv: Invitation, **out** s: User)

inv.to == r ; s := inv.from ; invites -= inv

other actions...

Example: concept Inviting

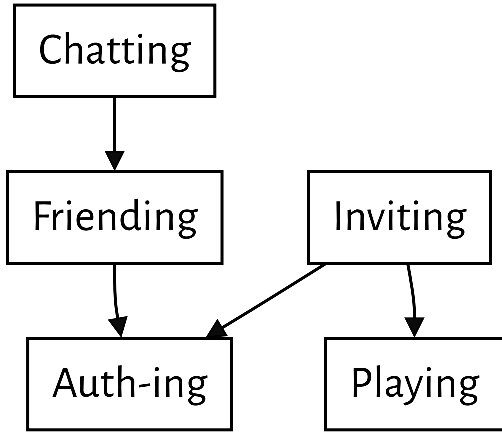
```
# concept Authenticating
__abbreviated from "Concept sync" tutorial!__
purpose: ... principle: ...
state:
    registered: set User
    username, password: registered -> one String
actions:
    register(un: String, pw: String, out user: User)
        registered += user ; user.username := un ; user.password := pw
    authenticate(un: String, pw: String, out user: User)
        require user.username == un and user.password == pw

# concept Sessioning [User]
__abbreviated from "Concept sync" tutorial!__
purpose: ... principle: ...
state:
    active: set Session
    user: active -> one User
actions:
    start(user: User, out session: Session)
        session.user := user
    getUser(session: Session, out user: User)
        user := session.user

# concept Friending [User]
purpose: ... principle: ...
state:
    friends: User -> set User
actions:
    friend(a: User, b: User)
        require (a, b) not in friends
        a.friends += b ; b.friends += a
    assertFriends(a: User, b: User)
        require (a, b) in friends

# concept Playing
purpose: ... principle: ...
state:
    ... Play ...
actions: ...
```

```
# concept Inviting [User, Event]
purpose: ... principle: ...
state:
    invites: set Invitation
    from, to: invites -> one User
    for: invites -> one Event
actions:
    invite(sender: User, recipient: User, event: Event, out inv: Invitation)
        invites += inv ; inv.from := sender ; inv.to := recipient
        inv.for := event
    accept(recipient: User, inv: Invitation, out sender: User, out event: Event
        require inv.to == recipient
        invites -= inv ; sender := inv.from
        event := inv.for

# app Puzzmo
include Authenticating as Auth
let User = Auth.User
include Sessioning [User]
include Playing
let Play = Playing.Play
include Friending [User]
include Inviting [User, none] as InvF
include Inviting [User, Play] as InvP

sync __inviteToFriend(from: User, to: User, out invite: InvF.Invitation)__
    InvF.invite(from, to, none, invite)

sync __friend(to: User, invite: InvF.Invitation)__
    InvF.accept(to, invite, from, none)
    Friending.friend(to, from)

sync __inviteToPlay(from: User, to: User, play: Play, out invite: InvP.Invitati
    Friending.assertFriends(from, to)
    InvP.invite(from, to, play, invite)

sync __joinGame(to: User, invite: InvP.Invitation)__
    InvP.accept(to, invite, from, play)
    Playing.SOMETHING(play)
```

Puzzmo

Dependencies

concepts in the app

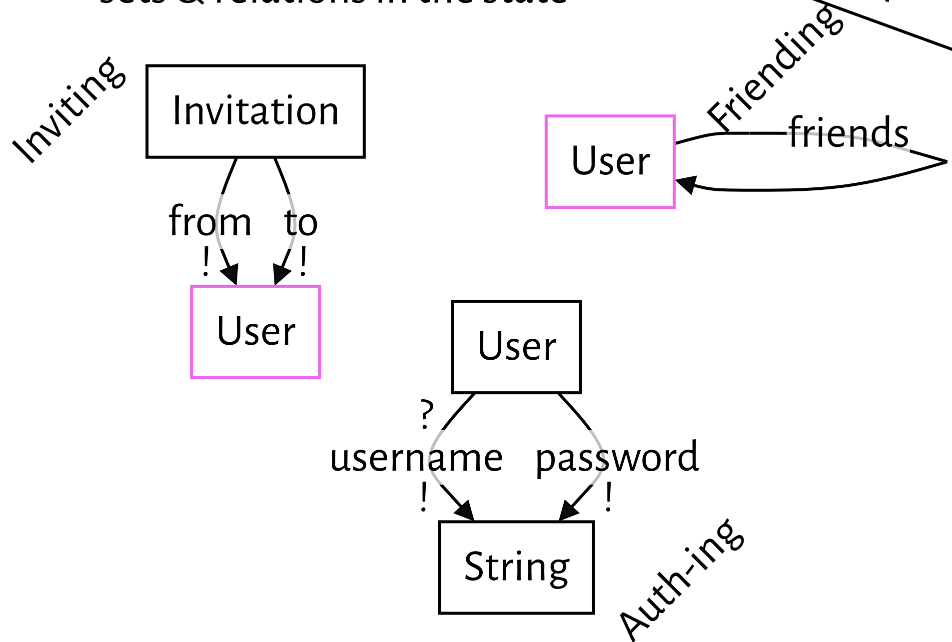


A → B *app including A must also include B*

Data Models

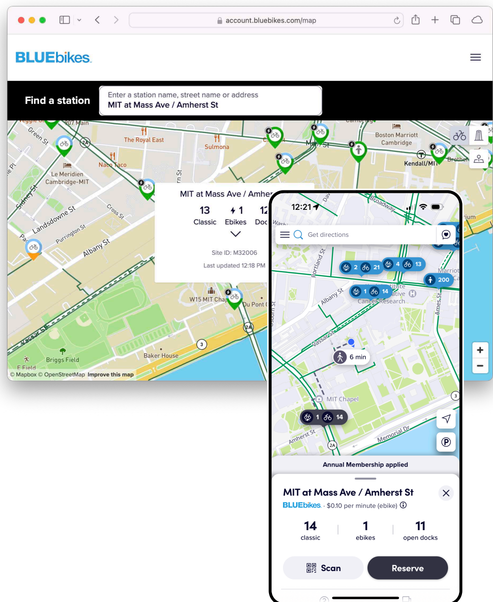
sets & relations in the state

which we can then compose into a global data model



T → U *relation from type T to type U* V *generic*

Client / Server

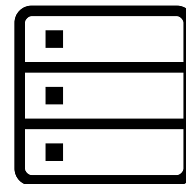


HTTP request

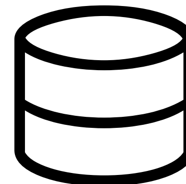
This interface requires design

› URLs › requests › responses

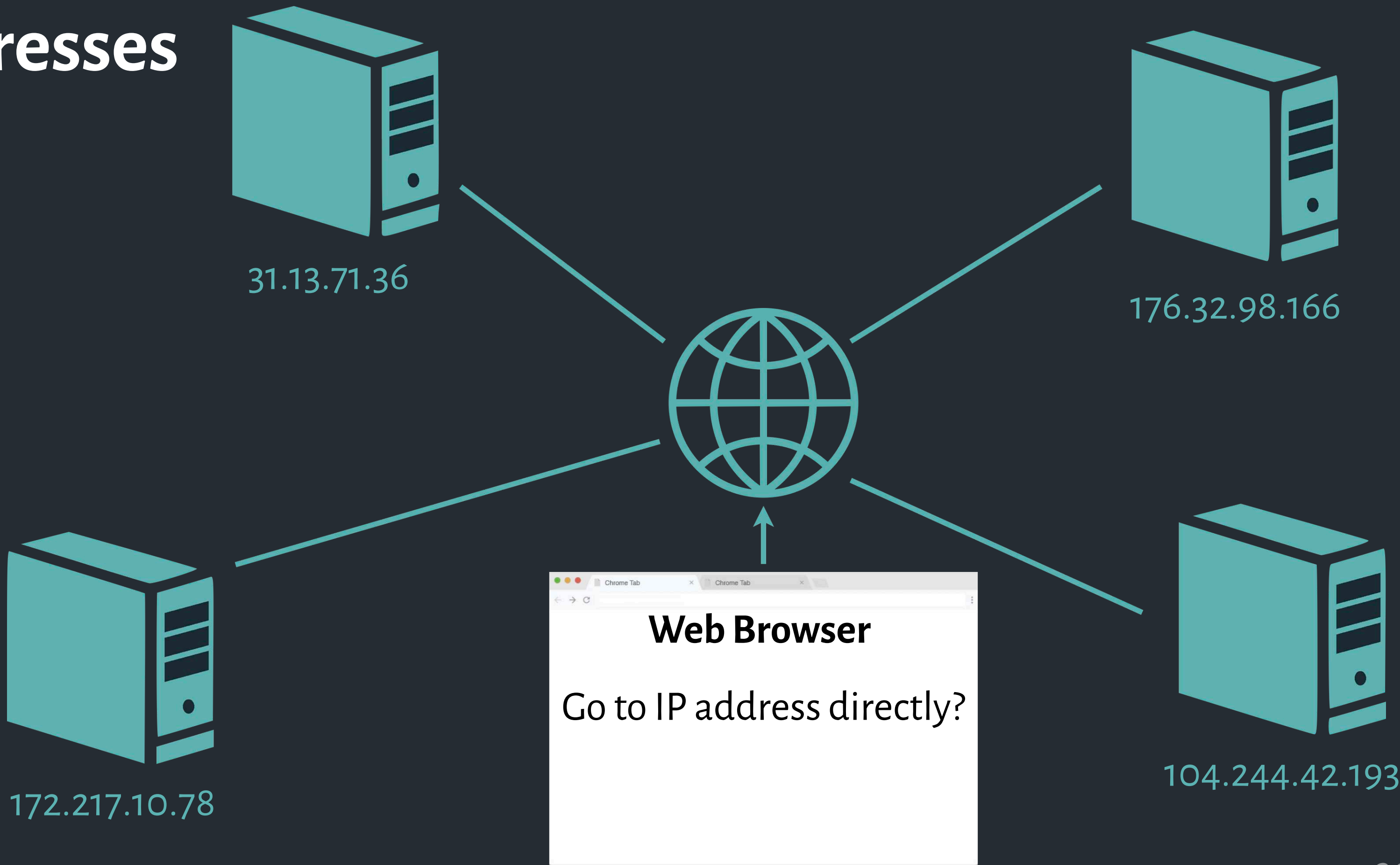
HTTP response



process request & build response



IP addresses



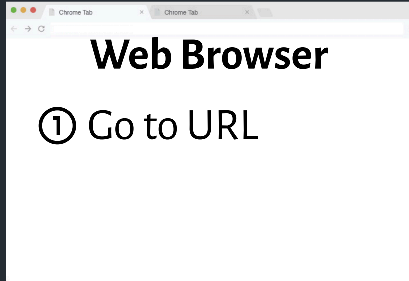
URLs: Uniform Resource Locators

protocol host path

https://61040-forum.csail.mit.edu/t/class-spotify-playlist/112?sort=score&status=all#footer-buttons

query fragment

Using a URL

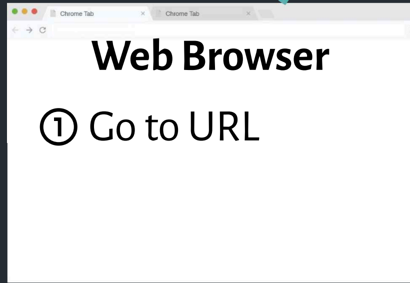


protocol host path query

https://61040-forum.csail.mit.edu/t/class-spotify-playlist/112?sort=score&status=all

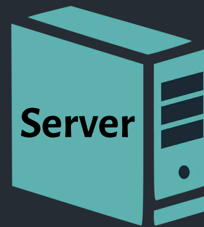
DNS lookup

② Look up domain name



③ Resolve to IP address

128.52.130.153



host

https://61040-forum.csail.mit.edu/t/class-spotify-playlist/112?sort=score&status=all

HTTP request

“The two hardest problems in CS are:
(i) cache invalidation,
(ii) naming things, and
(iii) off by one errors”

② Look up domain name

Web Browser

① Go to URL

⑧ Parse response & render page

③ Resolve to IP address

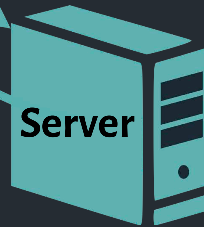
④ HTTP request

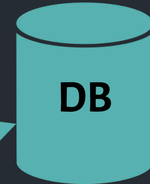
⑤ Route the request

⑥ Process request & build response

⑦ HTTP response

128.52.130.153





protocol host path query

https://61040-forum.csail.mit.edu/t/class-spotify-playlist/112?sort=score&status=all

HTTP methods

GET *url*

→ `GET` https://61040-forum.csail.mit.edu/top?order=posts
← `200` *<lots of html>*

POST *url* + *body* and PUT *url* + *body*

→ `POST` https://61040-forum.csail.mit.edu/bookmarks.json
body `{"bookmarkable_id": 148}`
← `200 {"success": "OK", "id": 7}`

DELETE *url*

→ `DELETE` https://61040-forum.csail.mit.edu/bookmarks/7.json
← `200 {"success": "OK"}`

… and several others

Early web service URLs

```
/shopping_cart.asp?action=update_qty&user=123
/postComment.jsp?entryID=853&text=...
/services.php?method=bid&item=236&...
```

- ✕ Inconsistent: different APIs might use different path and paramter conventions, and individual APIs might be internally inconsistent
- ✕ Difficult to maintain and extend (for developers)
- ✕ Not easily discoverable or recognizable (for users)

REST: Representational State Transfer

(Just some highlights*!*)

Client/server architecture

A protocol over HTTP defines the interface between client & server

Stateless

Server does not store state for individual clients: each request is self-contained

Representations

Resources are identified by URLs
Resources have representations in the protocol, transfered back and forth
Protocol reps are not necessarily stored reps (good old rep. independence)
Protocol reps include the information a client needs to make further requests,
e.g. to update an entity, find related entities, *etc.*

RESTful

"Applying verbs to nouns"

/people/arvind

Profile page: `/people/arvind.html`
Profile picture: `/people/arvind.jpg`
Data structure: `/people/arvind.json`

Collections:

`/people`
`/people/arvind/flair`

Instances:

`/people/arvind`
`/people/arvind/flair/275`

URL paths to identify resources (nouns)

- Use related paths to identify different representations
- Use hierarchy to indicate structure

RESTful

"Applying verbs to nouns"

Create: POST `/people/arvind/flair`
body `{ "type": "pin", ... }`

Read: GET `/people/arvind`

Update: PUT `/people/arvind/flair/5`
body `{ "text": "👋" }`

Delete: DELETE `/people/arvind/flair/10`

Profile page: `/people/arvind.html`
Profile picture: `/people/arvind.jpg`
Data structure: `/people/arvind.json`

Collections:
`/people`
`/people/arvind/flair`

Instances:
`/people/arvind`
`/people/arvind/flair/275`

URL paths to identify resources (nouns)

- Use related paths to identify different representations
- Use hierarchy to indicate structure

HTTP methods for different actions (verbs) on the resource

Consistency and data safety

mutation?

HTTP methods and data safety

URL paths to identify resources (nouns)
HTTP methods for different actions (verbs) on the resource

Create: POST `/people/arvind/flair`
Read: GET `/people/arvind`
Update: PUT `/people/arvind/flair/5`
Delete: DELETE `/people/arvind/flair/10`

Safe methods do not change the resource

Idempotent methods can be called multiple times with the same effect as calling once

Method	Safe	Idempotent
GET	✓	✓
POST	✕	✕
PUT	✕	✓
DELETE	✕	✓

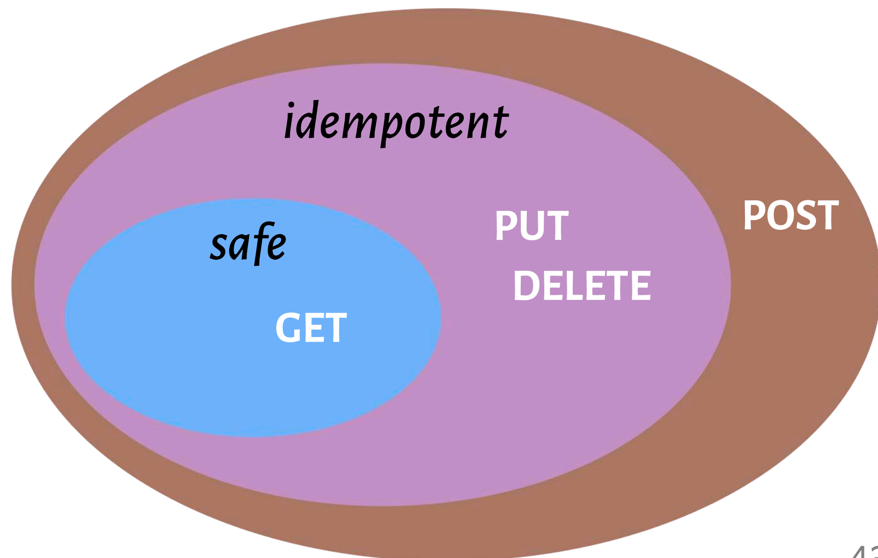
HTTP methods and data safety

URL paths to identify resources (nouns)
HTTP methods for different actions (verbs) on the resource

Create: POST `/people/arvind/flair`
Read: GET `/people/arvind`
Update: PUT `/people/arvind/flair/5`
Delete: DELETE `/people/arvind/flair/10`

Safe methods do not change the resource

Idempotent methods can be called multiple times with the same effect as calling once



Design a RESTful API for Puzzmo

What are the URL paths?

How are resources nested?

What are the HTTP methods?

www.puzzmo.com/play/flip-art/2xb2526t1

Puzzmo by Orta Therox & Zach Gage

Start multiplayer session

September 27 Daily Score: 3,132 00:10

Search users

Pause

Instructions Options

Flipart...

Lee ...

Inviting

Invite

ONLINE

Alice

Do you want to do the crossword puzzle?

Matti ...

Mark ...

Elliot ...

Puzzle Complete

Total solves: 159

Recover your 84 game streak (+7) by playing 6 more days in a row

Highscore Friends rank #2

3,132

Fewest rotations Friends rank #2

14 rotates

Fastest win Friends rank #1

00:10

14, Correct: 9/9

10

Next up: **Bigart**

Home

Play again

Shout your success!

View leaderboards

Friending

Friends Clubs

```
type Session = void;

export class Routes {

  // POST /friends/invitations/:userid
  inviteToFriend(sess: Session, to_user_id: string) { }

  // GET /friends/invitations
  listFriendInvitations(sess: Session) { }

  // POST /friends/:inviteid
  // -or- maybe better if it was a userid in the URL?
  makeFriend(sess: Session, invitation_id: string) { }

  // GET /friends
  allMyFriends(sess: Session) { }

  // POST /play/:playid/invitation/:userid
  inviteToPlay(sess: Session, play_id: string) { }

  // try this one in terms of the players? and with a userid?
  // PUT /play/:playid/players/:userid
  joinGame(sess: Session, invitation_id: string) { }
}
```

Design a RESTful API for Hacker News

What are the URL paths?

How are resources nested?

What are the HTTP methods?

Hacker News new | past | comments | ask | show | jobs | submit login

▲ Jackson structured programming (wikipedia.org)

106 points by haakonhr 63 days ago | hide | past | favorite | 69 comments

Posting

Upvoting

Favoriting

ew | past | comments | ask | show | jobs | submit login

▲ danielnicholas 63 days ago [–]

If you want an intro to JSP, you m … Jackson festschrift in 2009.

For those who don't know JSP, I'd

- There's a class of programming … addresses this class, b … only on i

- … ypal prob … d just recognizing them helps.

user: danielnicholas

created: 63 days ago

karma: 13

about:

submissions

comments

favorites

Karma Tracking

Commenting

X Developer Platform

Documentation

Twitter API v2 | Twitter API: Enterprise | Twitter API: Standard v1.1 | Twitter Ads API

Twitter API v2

Tweets

Bookmarks

- DELETE /2/users/:id/bookmarks/:tweet_id
- GET /2/users/:id/bookmarks
- POST /2/users/:id/bookmarks

Filtered stream

- GET /2/tweets/search/stream
- GET /2/tweets/search/stream/rules
- POST /2/tweets/search/stream/rules

Hide replies

- PUT /2/tweets/:id/hidden

Likes

- DELETE /2/users/:id/likes/:tweet_id
- DELETE /2/users/:id/likes/:tweet_id
- GET /2/tweets/:id/liking_users

tableau from Salesforce

Search

← Tableau Help > Tableau REST API Help > API Reference—All Methods

IN THIS ARTICLE

API Method Categories

The following table lists the Tableau Server REST API met
table also indicates which methods can be used with Table

- Analytics Extensions Settings Methods
- Ask Data Lens Methods
- Authentication Methods
- Connected App Methods
- Content Exploration Methods
- Dashboard Extensions Settings Methods
- Data Sources Methods
- Extract and Encryption Methods

qualtrics XM | Instructions | Guides | API Reference | SDKs | Search...

- Surveys CRUD API
  - Surveys
    - Get Survey GET
    - Update Survey PUT
    - Delete Survey DELETE
    - List Surveys GET
    - Import Survey POST
    - Share Survey POST
  - Survey Embedded Data Fields
  - Survey Quotas
  - Schemas
- Ticketing API
- Transaction Batches
- Users
  - Users
    - List Users GET
    - Create User POST
    - Get User GET
    - Delete User DELETE
    - Update User PUT
    - Who Am I GET
  - Users API Tokens
  - Schemas
- WhatsApp Distributions

Surveys CRUD API

Export

v3.0.0

API Base URL

Canadian Data Center: https://yul1.qualtrics.com/API/v3
Washington, DC Area Data Center (previously CO1): https://iad1.qualtri
San Jose, California Data Center (previously AZ): https://sjc1.qualtrics.
European Union Data Center (previously EU2 or EU): https://fra1.qualtri
London, United Kingdom Data Center: https://lhr1.qualtrics.com/API/v3
Sydney, Australia Data Center (previously AU1): https://syd1.qualtrics.c
Singapore Data Center: https://sin1.qualtrics.com/API/v3
Tokyo, Japan Data Center: https://hnd1.qualtrics.com/API/v3
US Government Data Center: https://gov1.qualtrics.com/API/v3
Mock Server: https://stoplight.io/mocks/qualtricsv2/publicapidocs/60937

Security

API Key

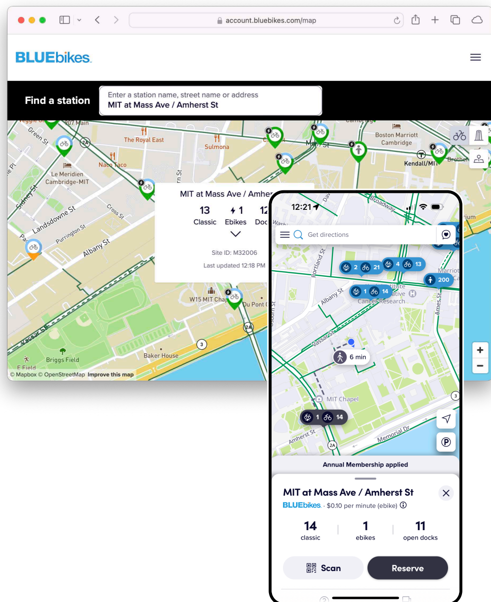
This is a schema for x-api-token header authentication.

An API key is a token that you provide when making API calls. Include the token in a header parameter called `X-API-TOKEN`.

Example: `X-API-TOKEN: 123`

OAuth 2.0

Client / Server

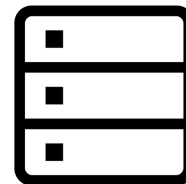


HTTP request

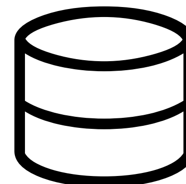
This interface requires design

› URLs › requests › responses

HTTP response



process request
& build response



Node.js and Express.js

JavaScript is *(as you know)* single-threaded

At the core of the JS runtime is the ***event queue***, a first-in-first-out queue that stores events that have arrived from various sources, including:

- HTTP requests
- File I/O
- Timers

This queue is serviced by an ***event loop*** that repeatedly checks the queue for the next event and calls the event's associated handler function

While our code is running, the event loop is not processing events! It will not run until control returns to the runtime by...

- returning to the top of call stack
- giving up control in a (non-first) `await`

Dealing with asynchrony

Callbacks and ***Promises*** are our tools for asynchronous computation

Expect to use Promises and `async` / `await` extensively

To review, please revisit the 6.102 readings at web.mit.edu/6.102/

14: *Concurrency*	gives an overview of concurrency
15: *Promises*	introduces Promises and `async` / `await`
16: *Mutual Exclusion*	discusses hazards like race conditions and deadlock
17: *Callbacks & GUIs*	discusses callbacks and the event loop
18: *Message-Passing & Networking*	introduces the basics of client/server web applications

Today

RESTful API design

- HTTP
- URLs
- Data safety

Preparing for implementation in Node.js

Looking ahead

Reactive frameworks
User interface design