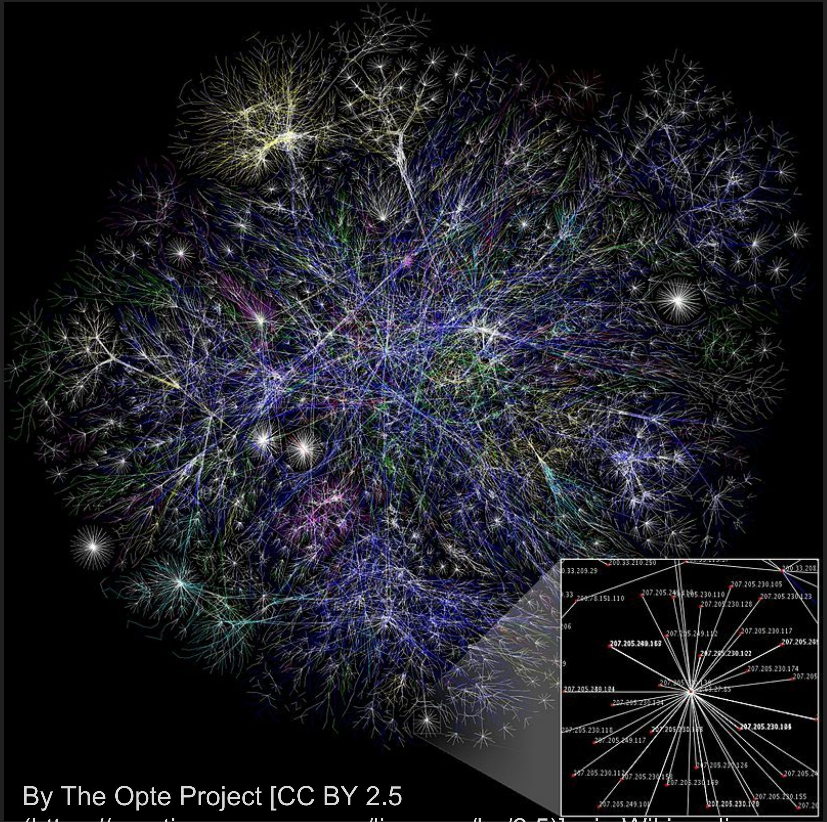
REST APIs

A Crash Course

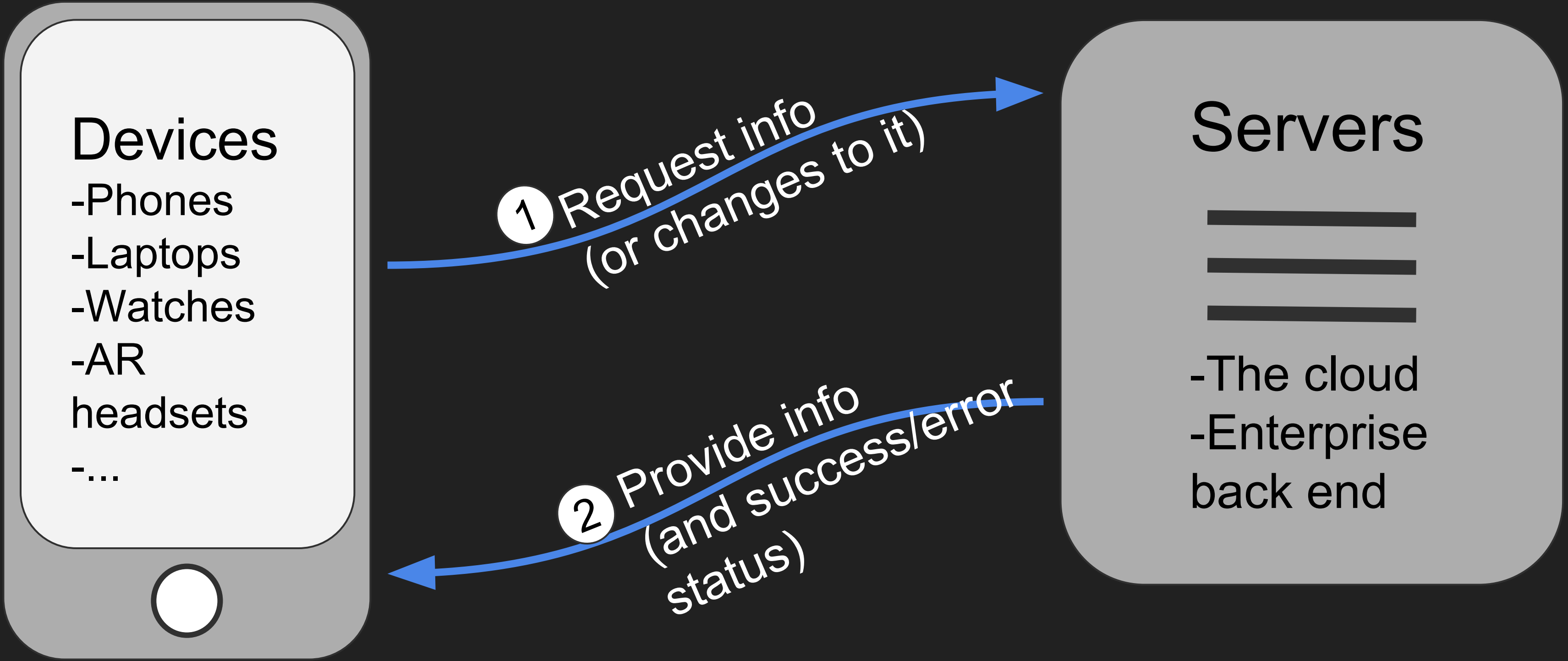
Andrew Puglionesi

Developers need the internet

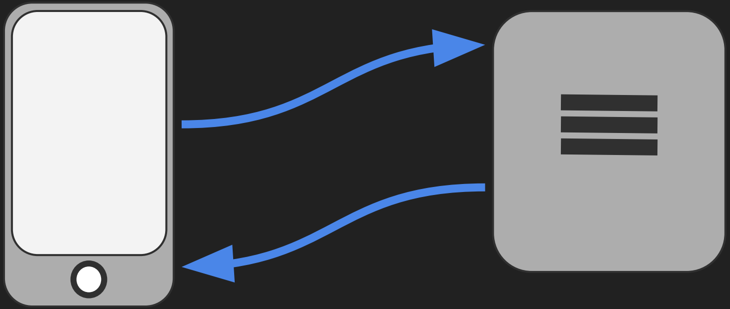
- The internet is unavoidable
- “But I’m not a web developer. I want to work on…”
  - Mobile applications
  - Video games
  - … Microsoft Word?
- Sorry, pal


By The Opte Project [CC BY 2.5 (https://creativecommons.org/licenses/by/2.5)], via Wikimedia Commons

A Simplified View



What kind of “info”?



It’s not key, but often JSON data

```
{
      “userID”:1,
      "firstName":"Alan",
      "lastName":"Turing",
      “birthday":"1912-06-23T00:00:00.000Z",
      "hobbies": [
        "Turing machines",
        "Marathons"
      ]
}
```

REST (Representational State Transfer)

- REST is an architectural style for communication over the internet
- It was designed by a PhD student, Roy T. Fielding, at UCI in 2000
- It is immensely popular

REST Constraints

- REST is **not** a language, a framework, or a tool
- REST is a concept
    - Client-server architecture: Separates the “UI concerns” from the “data storage concerns”
    - Stateless: It’s up to the client-side to store and provide all context in its requests to the server (which thus doesn’t *save state* for the client)
    - The server’s response must say whether the data it sends the client can be cached
    - Uniform interface (more to come)
    - Layered System—there might be a group/chain of servers involved

REST... for developers

- Front end interacts with “resources,” which are just data on the server

For a REST API, HTTP methods should generally be used as follows:

- GET: request data from the server (an entire collection or by ID)
- PUT: request modification of data by the server
- DELETE: Request deletion of data by the server
- POST: request creation of data by the server
- POST often serves as a catch-all for anything not listed

REST doesn’t entail a particular format (e.g., JSON, XML) for data being sent back and forth

HTTP methods + URIs are descriptive

GET

- mysite.com/users/45
  - Retrieve info about user #45
- mysite.com/users
  - Retrieve a collection of users

PUT

- mysite.com/users/45
  - Modify data for user #45

POST

- mysite.com/users
  - Create a new user

DELETE

- mysite.com/users/45
  - Delete user #45

Informative Error Status

(There are a lot more HTTP errors than 404, just FYI)

```
# GET at /students/<id> (Django REST Framework example)
class StudentInstance(APIView):
  # retrieve an existing student with the id passed in with the URL
  def get(self, request, id):
    # grab student from DB
     student = Student.objects.filter(id=id).first()
     if not student:
        return Response(status=status.HTTP_404_NOT_FOUND,
               data={'message':’No student with that ID’})
     # parse Student object to JSON
     serializer = StudentSerializer(student)
     return Response(serializer.data)
```

Further Reading

My starter/demo code for the Django REST API:
github.com/**aop4**/heroku-django-REST-template

Roy Fielding’s Dissertation (fairly readable):
https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

A quick (and useful) overview of REST:
https://www.restapitutorial.com