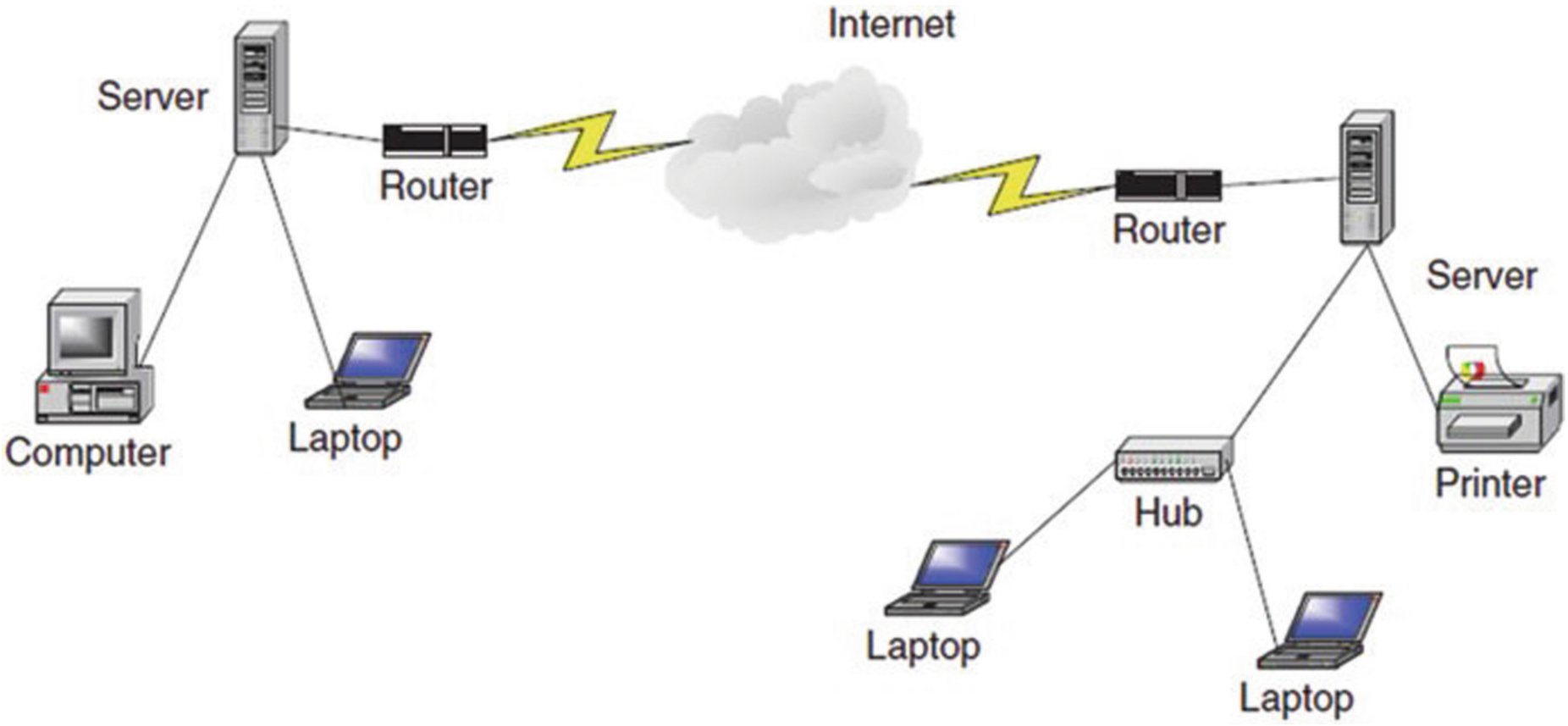
Optics for servers

What is a server?

What is a lens?

What’s the library like?

What is a web server?


Internet
Server
Router
Router
Server
Computer
Laptop
Printer
Hub
Laptop
Laptop

Internet protocol suite

Application layer

BGP · DHCP(v6) · DNS · FTP · **HTTP** · HTTPS · IMAP · LDAP · MGCP · MQTT · NNTP · NTP · POP · PTP · ONC/RPC · RTP · RTSP · RIP · SIP · SMTP · SNMP · SSH · Telnet · TLS/SSL · XMPP · *more...*

Transport layer

TCP · UDP · DCCP · SCTP · RSVP · *more...*

Internet layer

IP (IPv4 · IPv6) · ICMP(v6) · ECN · IGMP · IPsec · *more...*

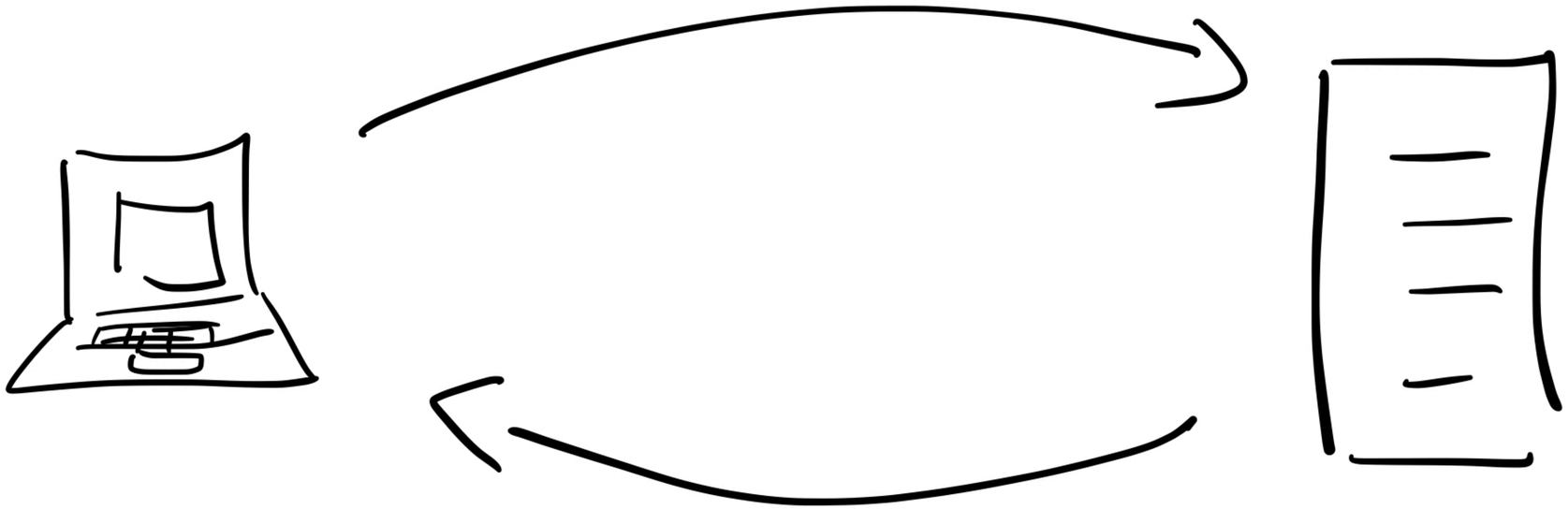
Link layer

ARP · NDP · OSPF · Tunnels (L2TP) · PPP · MAC (Ethernet · Wi-Fi · DSL · ISDN · FDDI) *more...*

GET /software/htp/cics/index.html HTTP/1.1

Request method ⬍
GET
HEAD
POST
PUT
DELETE
CONNECT
OPTIONS
TRACE
PATCH

GET /resource
POST / user
PUT / data



ok ...

Endpoint

API

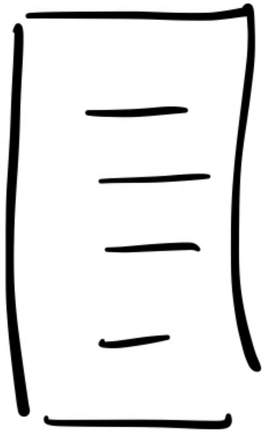
GET /path

POST /to

PUT /resource

url path

Server



NodeJS/ember

Swagger

Servant

```
var http = require('http'); // Import Node.js core module

var server = http.createServer(function (req, res) {   //create web server
    if (req.url == '/') { //check the URL of the current request

        // set response header
        res.writeHead(200, { 'Content-Type': 'text/html' });

        // set response content
        res.write('<html><body><p>This is home Page.</p></body></html>');
        res.end();

    }
    else if (req.url == "/student") {

        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('<html><body><p>This is student Page.</p></body></html>');
        res.end();

    }
    else if (req.url == "/admin") {

        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('<html><body><p>This is admin Page.</p></body></html>');
        res.end();

    }
    else
        res.end('Invalid Request!');

});

server.listen(5000); //6 - listen for any incoming requests

console.log('Node.js web server at port 5000 is running..')
```

Source: https://www.tutorialsteacher.com/nodejs/create-nodejs-web-server

Parsing

Parsing

Parsing

```
var http = require('http'); // Import Node.js core module

var server = http.createServer(function (req, res) {   //create web server
    if (req.url == '/') { //check the URL of the current request

        // set response header
        res.writeHead(200, { 'Content-Type': 'text/html' });

        // set response content
        res.write('<html><body><p>This is home Page.</p></body></html>');
        res.end();

    }
    else if (req.url == "/student") {

        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('<html><body><p>This is student Page.</p></body></html>');
        res.end();

    }
    else if (req.url == "/admin") {

        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('<html><body><p>This is admin Page.</p></body></html>');
        res.end();

    }
    else
        res.end('Invalid Request!');

});

server.listen(5000); //6 - listen for any incoming requests

console.log('Node.js web server at port 5000 is running..')
```

Source: https://www.tutorialsteacher.com/nodejs/create-nodejs-web-server

```
var http = require('http'); // Import Node.js core module

var server = http.createServer(function (req, res) {   //create web server
    if (req.url == '/') { //check the URL of the current request

        // set response header
        res.writeHead(200, { 'Content-Type': 'text/html' });

        // set response content
        res.write('<html><body><p>This is home Page.</p></body></html>');
        res.end();

    }
    else if (req.url == "/student") {

        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('<html><body><p>This is student Page.</p></body></html>');
        res.end();

    }
    else if (req.url == "/admin") {

        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('<html><body><p>This is admin Page.</p></body></html>');
        res.end();

    }
    else
        res.end('Invalid Request!');

});

server.listen(5000); //6 - listen for any incoming requests

console.log('Node.js web server at port 5000 is running..')
```

Responding

Responding

Responding

Source: https://www.tutorialsteacher.com/nodejs/create-nodejs-web-server

```
var http = require('http'); // Import Node.js core module

var server = http.createServer(function (req, res) {   //create web server
    if (req.url == '/') { //check the URL of the current request

        // set response header
        res.writeHead(200, { 'Content-Type': 'text/html' });

        // set response content
        res.write('<html><body><p>This is home Page.</p></body></html>');
        res.end();

    }
    else if (req.url == "/student") {

        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('<html><body><p>This is student Page.</p></body></html>');
        res.end();

    }
    else if (req.url == "/admin") {

        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('<html><body><p>This is admin Page.</p></body></html>');
        res.end();

    }
    else
        res.end('Invalid Request!');

});

server.listen(5000); //6 - listen for any incoming requests

console.log('Node.js web server at port 5000 is running..')
```

Output

Output

Output

Source: https://www.tutorialsteacher.com/nodejs/create-nodejs-web-server

API is a consequence of the implementation

Error prone

No possibility for programming endpoints

Info

Tags

Search

Device

GET /devices

POST /devices

Z-Wave

POST /lighting/dimmers/{deviceId

POST /lighting/dimmers/{deviceId

GET /lighting/switches/{deviceId

POST /lighting/switches/{deviceId

POST /lighting/switches/{deviceId

GET /lightingSummary

Environment

GET /temperature

GET /temperature/forecast/{days

GET /temperature/{zoneId}

GET /temperature/{zoneId}/heate

POST /temperature/{zoneId}/heate

Zones

GET /zones

GET /zones/{zoneId}/quiet

Models

MODEL LightingSummary

MODEL LightingZone

MODEL LightingZoneStatus

Aa SAVE SYNC

```yaml
swagger: '2.0'
info:
  version: 1.0.0
  title: home-iot-api
  description: The API for the EatBacon IOT project

consumes:
  - application/json
produces:
  - application/json
paths:
  /devices:
    get:
      tags:
      - Device
      description: returns all registered devices
      operationId: getDevices
      parameters:
      - in: query
        name: skip
        type: integer
        format: int32
        description: number of records to skip
      - in: query
        name: limit
        type: integer
        format: int32
        description: max number of records to return
      responses:
        200:
          description: All the devices
          schema:
            type: array
            items:
              type: string
              format: uri
              example: http://10.0.0.225:8080
    post:
      tags:
      - Device
      operationId: register
      parameters:
      - in: body
```

Last Saved: 1:39:10 am - Jun 24, 2021

VALID

[Base URL: virtserver.swaggerhub.com/andrevideta/demo-api/1.0.0]

The API for the EatBacon IOT project

Schemes

HTTPS

Device

GET /devices

POST /devices

Z-Wave

POST /lighting/dimmers/{deviceId}/{value}

POST /lighting/dimmers/{deviceId}/{value}/timer/{timeunit}

GET /lighting/switches/{deviceId}

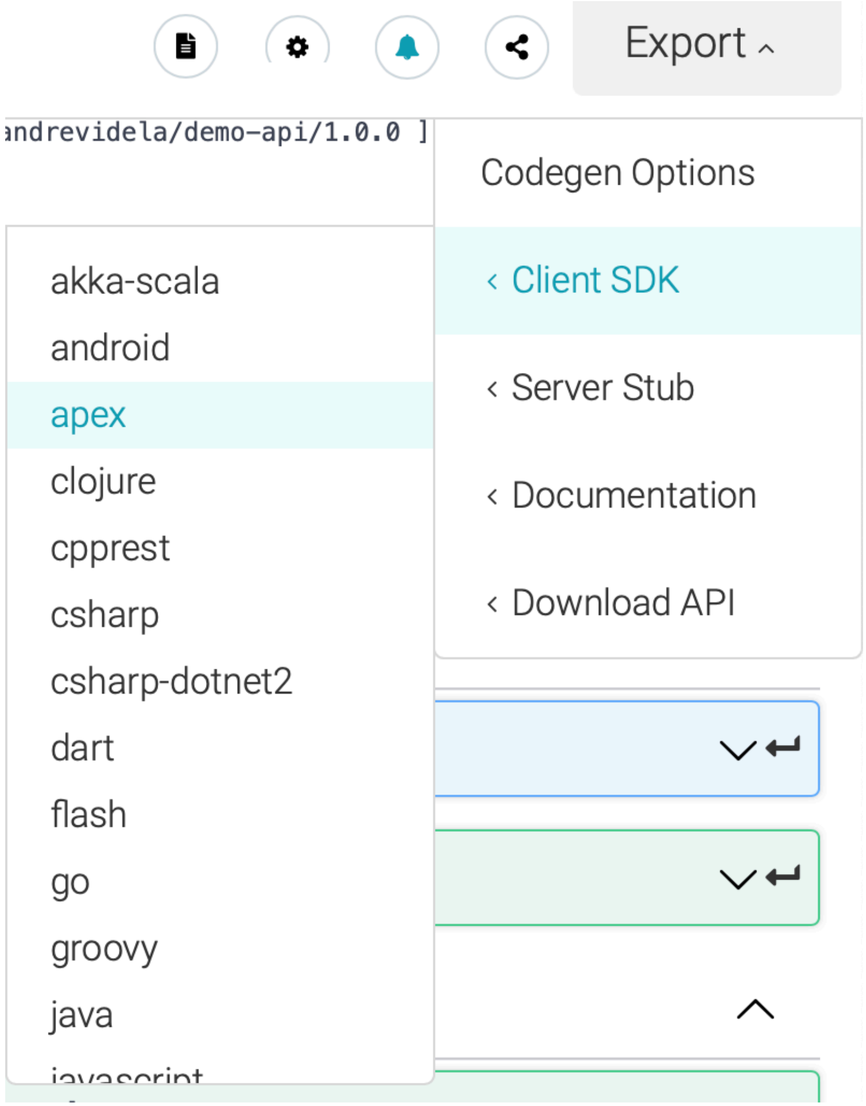
POST /lighting/switches/{deviceId}/{value}

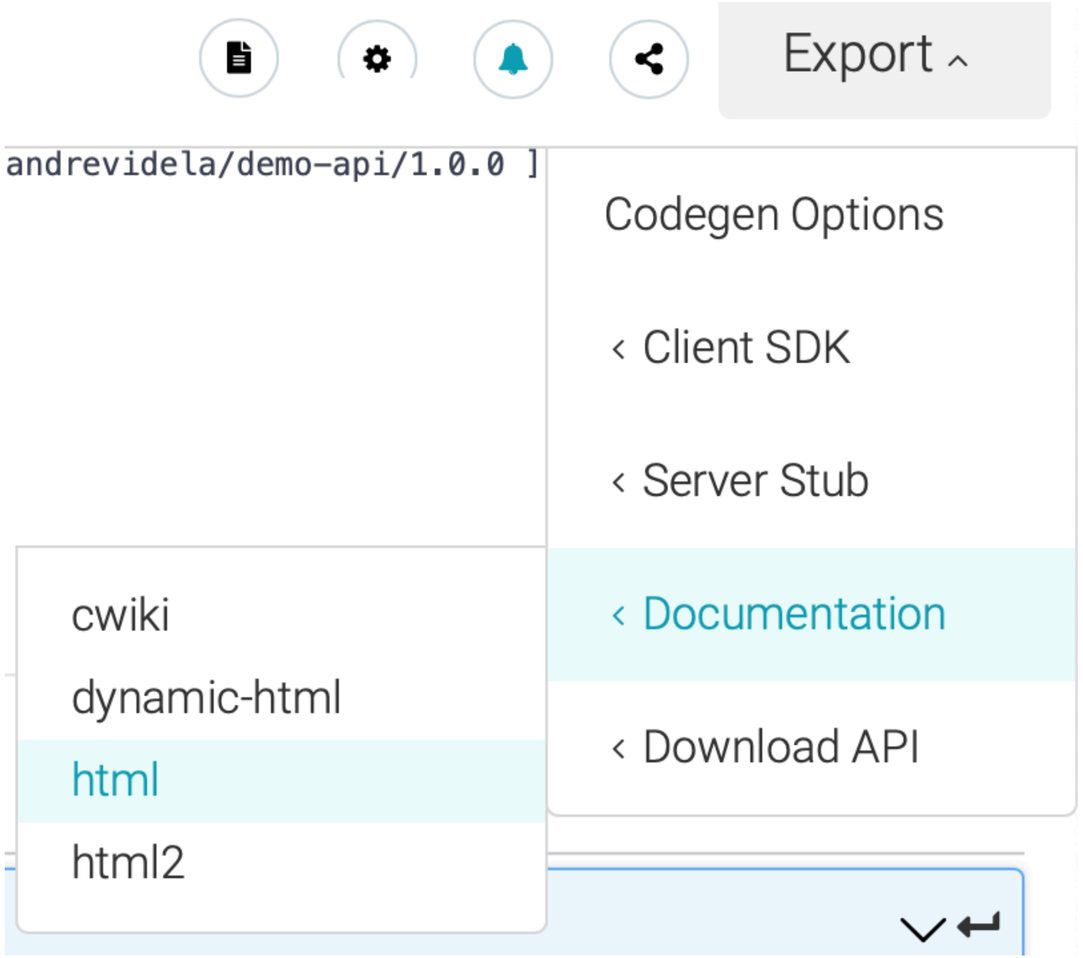
POST /lighting/switches/{deviceId}/{value}/timer/{minutes}

GET /lightingSummary

Environment

GET /temperature


Export
andrevidela/demo-api/1.0.0]
Codegen Options
Client SDK
Server Stub
Documentation
Download API
akka-scala
android
apex
clojure
cpprest
csharp
csharp-dotnet2
dart
flash
go
groovy
java


Export
andrevidela/demo-api/1.0.0]
Codegen Options
Client SDK
Server Stub
Documentation
Download API
cwiki
dynamic-html
html
html2

Additional tool in the workflow

Source of truth outside of the code

No programming

```haskell
type API = "position" :> Capture "x" Int :> Capture "y" Int :> Get '[JSON] Position
      :<|> "hello" :> QueryParam "name" String :> Get '[JSON] HelloMessage
      :<|> "marketing" :> ReqBody '[JSON] ClientInfo :> Post '[JSON] Email
```

```haskell
server3 :: Server API
server3 = position
     :<|> hello
     :<|> marketing

  where position :: Int -> Int -> Handler Position
        position x y = return (Position x y)

        hello :: Maybe String -> Handler HelloMessage
        hello mname = return . HelloMessage $ case mname of
          Nothing -> "Hello, anonymous coward"
          Just n  -> "Hello, " ++ n

        marketing :: ClientInfo -> Handler Email
        marketing clientinfo = return (emailForClient clientinfo)
```

API and code always in sync

API and documentation always in sync

Everything lives in the same language

	NodeJS	Swagger	Servant
Sync with implementation	✅	✅	✅
Sync with docmentation	❌	✅	✅
Easy to extend	❌	❌	✅

Servant is perfect ?

APIs are kinds

```
type SoundcloudTrackAPI = "tracks" :>
  (     QueryParam "client_id" T.Text
      :> QueryParams "genres" T.Text
      -> Get [JSON] [ST.Track]
  :<|> QueryParam "client_id" T.Text
      :> Capture "id" Int
      :> Get [JSON] ST.Track
  )
```

```
• Couldn't match type ‘Client
                         (QueryParam "client_id" T.Text :> QueryParams "genres"
T.Text
                          -> Get '[JSON] [ST.Track]
                             :<|> (QueryParam "client_id" T.Text
                                   :> (Capture "id" Int :> Get '[JSON] ST.Track)))’
                  with ‘(Maybe T.Text
                         -> [T.Text] -> Manager -> BaseUrl -> ClientM [ST.Track])
                        :<|> (Maybe T.Text
                             -> Int -> Manager -> BaseUrl -> ClientM ST.Track)’
  Expected type: (Maybe T.Text
                  -> [T.Text] -> Manager -> BaseUrl -> ClientM [ST.Track])
                 :<|> (Maybe T.Text
                       -> Int -> Manager -> BaseUrl -> ClientM ST.Track)
    Actual type: Client SoundcloudTrackAPI
• In the expression: client soundcloudAPI
  In a pattern binding:
    (searchTracksByGenre :<|> getTrack) = client soundcloudAPI
```

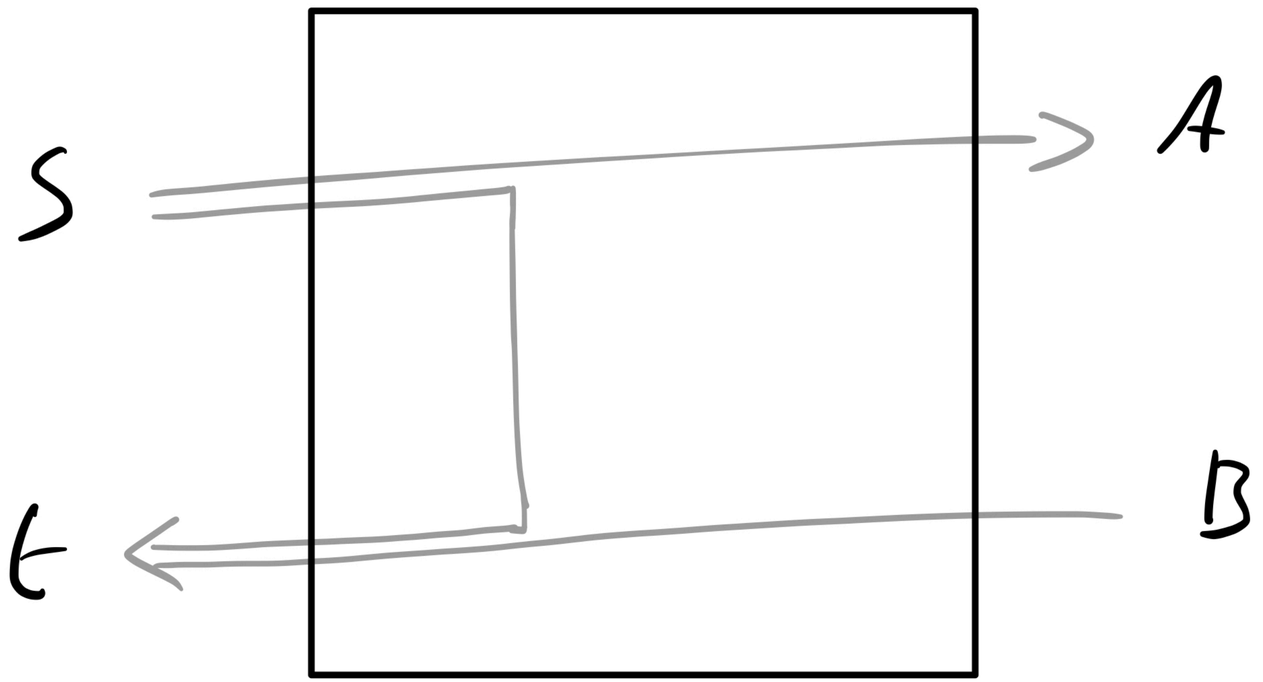
Types are not first class

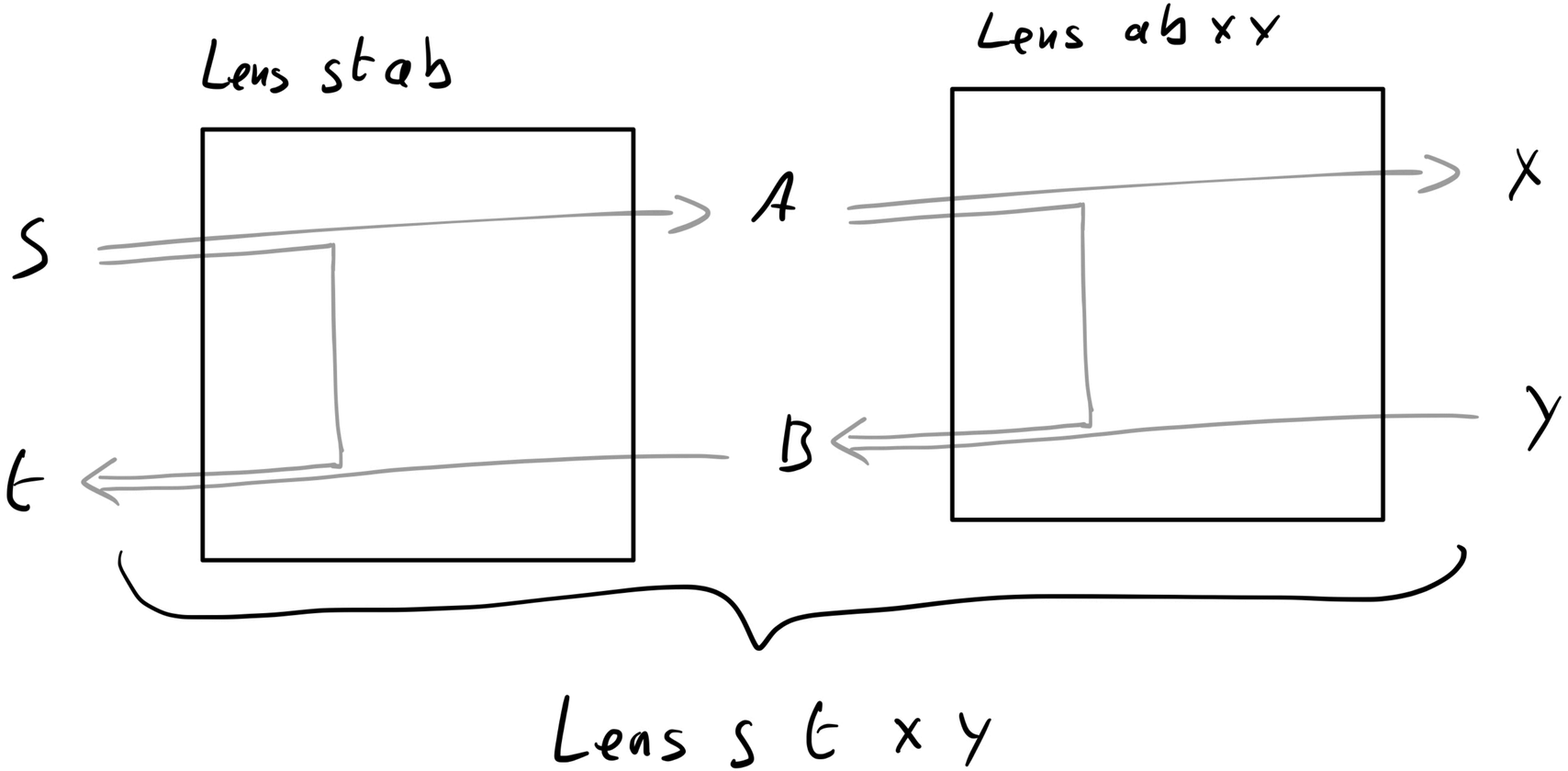
Comparing two APIs? ❌

Generating an API from the runtime? ❌

Compose Servers? ❌

What is a lens?


S
A
t
B


Lens s t a b
Lens a b x y
s
t
A
B
x
y
Lens s t x y

```
record Lens (a, b, s, t : Type)
where
  constructor MkLens
  get : s -> a
  set : s :*: b -> t
```

Server as lenses

```
GET /:user/todo/all                    User -> List Todo

POST /:user/todo/new                   User -> Todo -> ()
content-type: JSON
body: { “title” : “string”,
        “body” : “string”}
```

```
User -> List Todo                    s -> a

User -> Todo -> ()                   s -> b -> t
```

State management

```
GET /:user/todo/all
```

: Query the state

```
POST /:user/todo/new
content-type: JSON
body: { “title” : “string”,
        “body” : “string”}
```

: Update the state

```
GET /:user/todo/all                User -> State -> List Todo

POST /:user/todo/new               User -> State -> Todo —> State
content-type: JSON
body: { “title” : “string”,
        “body” : “string”}
```

```
(arg, st) -> output                          s -> a

(arg, st) -> update —> (change, st)     s -> b —> t
```

```
                    (s,st) -> a
st : Type ->
                    (s,st) -> b -> (t, st)
```

```
GET /:user/todo/all

POST /:user/todo/new
content-type: JSON
body: { “title” : “string”,
        “body” : “string”}
```

```
(User, Map User (List Todo)) -> Todo

(User, Map User (List Todo)) -> Todo ->
                 ((), Map User (List Todo))
```

DEMO

Big WIP, lots of rough edges

Not production ready

Todo

- Support HTTP requests (LOL)
- Generate client code
- Support get-only extensions
- Support parameterised lenses
- Purely type-directed API description
- Dependently-typed APIs
- Ensure no API overlapping
- Error management
- Expose content type

- Session types? UDP? Bluetooth? API Visibility?
- Databases as lenses? Microservices?
- Server as co-data? Generalising to all interactive processes?
- Categorical semantics? Containers? Dependent lenses?

Thank you