COMSE6998: Modern Serverless Cloud Applications

Lecture 4: Lecture 4: TLDS, Swagger, Pagination, RESTAngular, Programmable Web, 2nd Project

Dr. Donald F. Ferguson
Donald.F.Ferguson@gmail.com

Contents

- Introduction
  - TAs
  - Project Definitions
  - Q&A
- ~~Evolution and Motivation: Web apps → SOA/Microservices → Serverless~~
- Top-Level Design Specification
- New technical topics
  - Swagger
  - ~~Pagination~~
  - ~~RESTAngular (Overview)~~
  - Programmable Web
- Second Project: Using Web APIs.

Introduction

TAs

- Kushwanth Ram Kesarla Shantharam
  - Just assigned.
  - Should be attending class today.

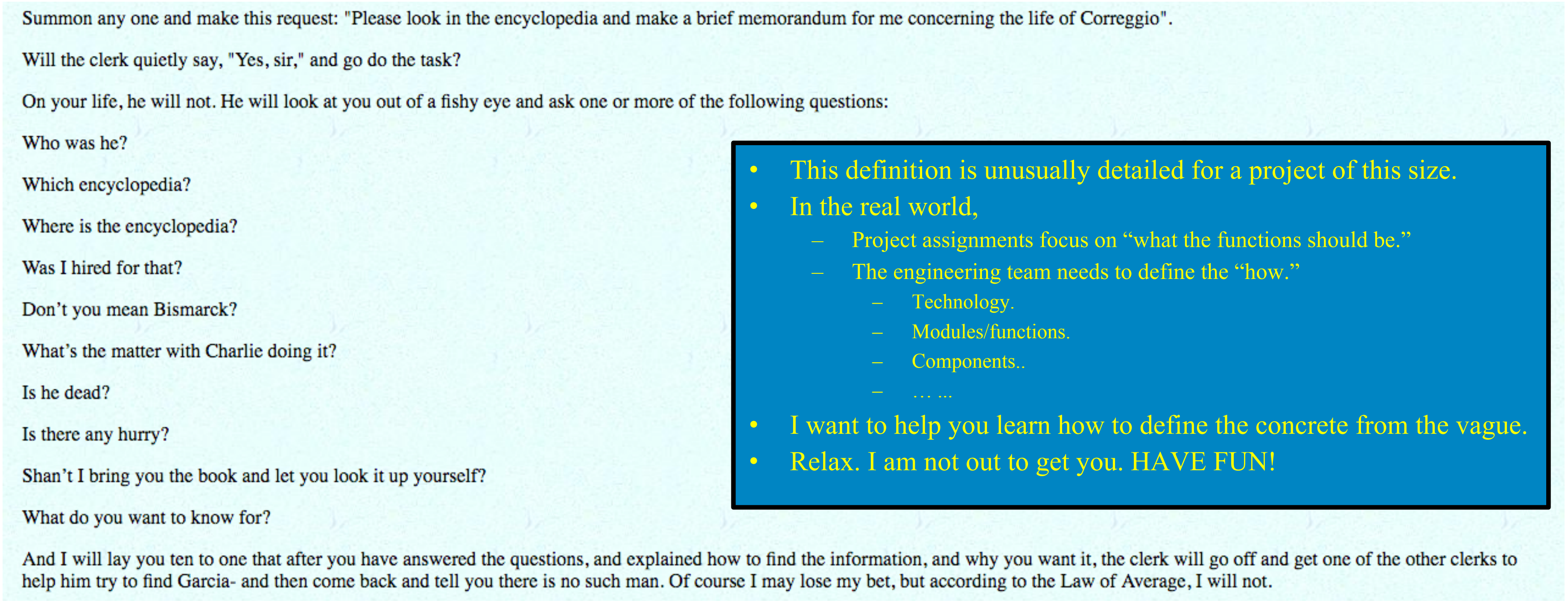
- Jingxiao Gu
  - Attends class.
  - Has announced office hours.
  - Active on Piazza.

- One more to be named, but have a good candidate.

First Project Requirements (from Lecture 2)

- Define DynamoDB Tables
  - Customer (lastname, firstname, email, phone number, address_ref)
  - Address (UUID, city, street, number, zip code)
- Implement a Lambda function for Customer and Address, e.g. for Customer
  - Methods
    - GET – key is “email.”
    - POST (Create)
      - Body is the data, but …
      - Cannot create (POST) if there is already a customer with that email.
    - PUT (Update)
      - Body is a subset of the JSON fields.
      - Update only those fields.
      - Cannot update an object that does not exist.
    - DELETE – key is email.
  - For all function, implement validation checks, e.g. no ”new fields,” zipcode is a number with 5 digits, …
- API Gateway
  - Define resources /Customers and /Addresses
  - POST on /Customers and /Addresses
  - GET, PUT and DELETE on /Customers/{email} and /Addresses/{id}
  - Navigation works /Customers/{email}/address returns the address.

Think, and “Carry a Message to Garcia” (https://en.wikipedia.org/wiki/A_Message_to_Garcia)



Summon any one and make this request: "Please look in the encyclopedia and make a brief memorandum for me concerning the life of Correggio".

Will the clerk quietly say, "Yes, sir," and go do the task?

On your life, he will not. He will look at you out of a fishy eye and ask one or more of the following questions:

Who was he?

Which encyclopedia?

Where is the encyclopedia?

Was I hired for that?

Don’t you mean Bismarck?

What’s the matter with Charlie doing it?

Is he dead?

Is there any hurry?

Shan’t I bring you the book and let you look it up yourself?

What do you want to know for?

And I will lay you ten to one that after you have answered the questions, and explained how to find the information, and why you want it, the clerk will go off and get one of the other clerks to help him try to find Garcia- and then come back and tell you there is no such man. Of course I may lose my bet, but according to the Law of Average, I will not.

- This definition is unusually detailed for a project of this size.
- In the real world,
  - Project assignments focus on “what the functions should be.”
  - The engineering team needs to define the “how.”
    - Technology.
    - Modules/functions.
    - Components..
    - … ...
- I want to help you learn how to define the concrete from the vague.
- Relax. I am not out to get you. HAVE FUN!

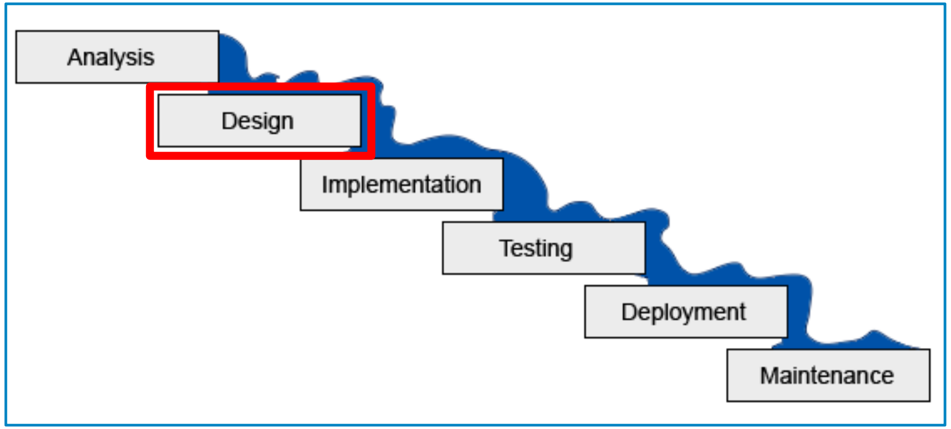
Think. Try. You will make mistakes. Correcting mistakes is how we learn.

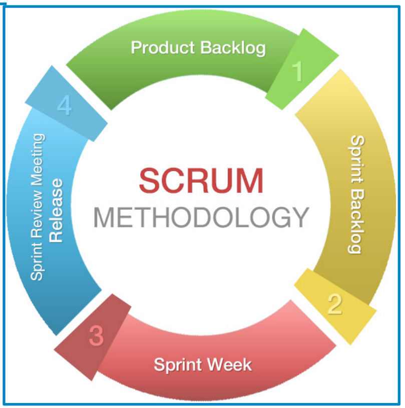
Q&A

Top-Level Design Specification

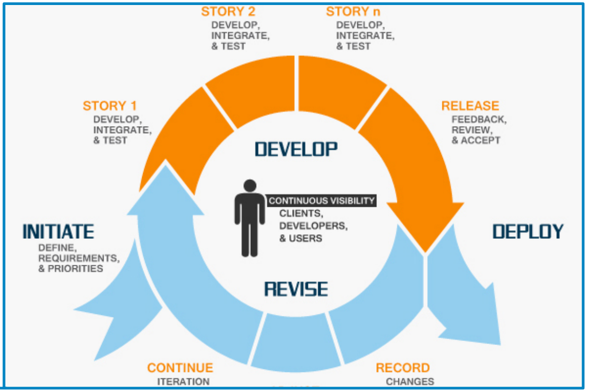
Agile Development

Traditional "Waterfall" Methodology





Design seems to be missing?



STORY 2
DEVELOP, INTEGRATE, & TEST
STORY n
DEVELOP, INTEGRATE, & TEST
STORY 1
DEVELOP, INTEGRATE, & TEST
RELEASE
FEEDBACK, REVIEW, & ACCEPT
DEVELOP
CONTINUOUS VISIBILITY
CLIENTS, DEVELOPERS, & USERS
INITIATE
DEFINE, REQUIREMENTS, & PRIORITIES
DEPLOY
REVISE
CONTINUE
ITERATION
RECORD
CHANGES

Observations

Weinberg's Second Law: "If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization."

Sr. Celestina: "If you cannot write it down, you do not understand it."

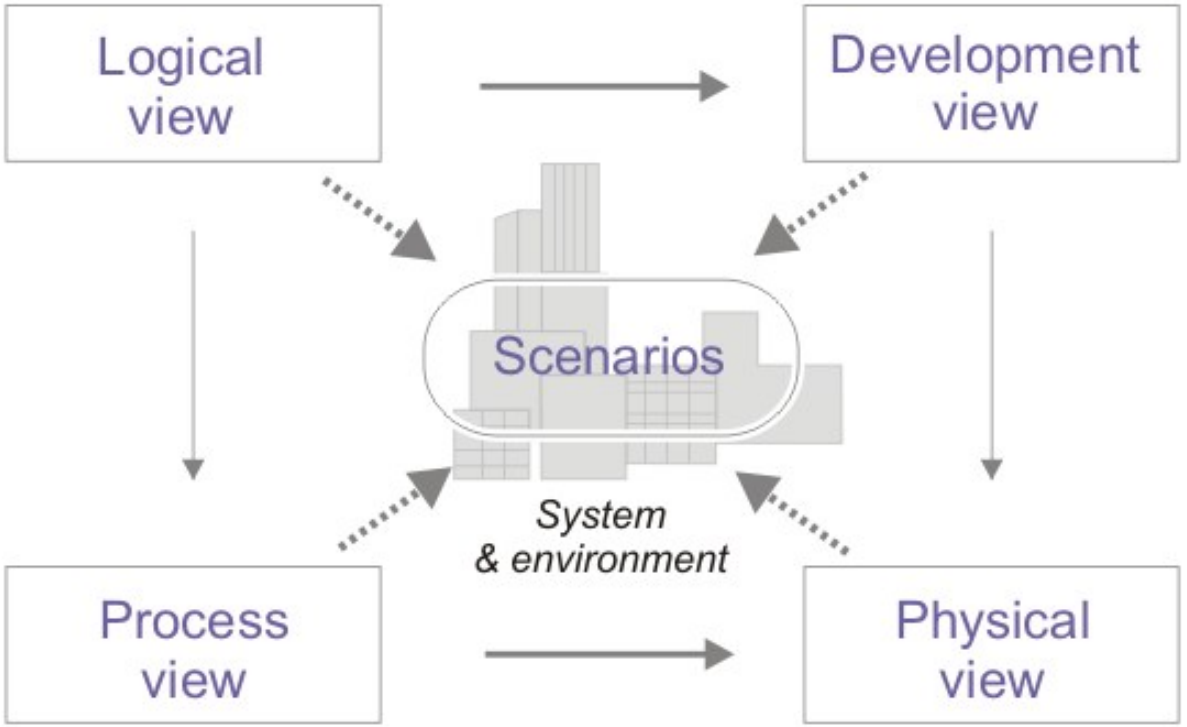
"it is a custom
More honor'd in the breach than the observance"
Hamlet Act 1, scene 4, 7–16

Agile Manifesto (http://agilemanifesto.org/):

- Individuals and interactions over processes and tools
- Working software over **comprehensive** documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

4+1 architectural view model

http://en.wikipedia.org/wiki/4%2B1_architectural_view_model

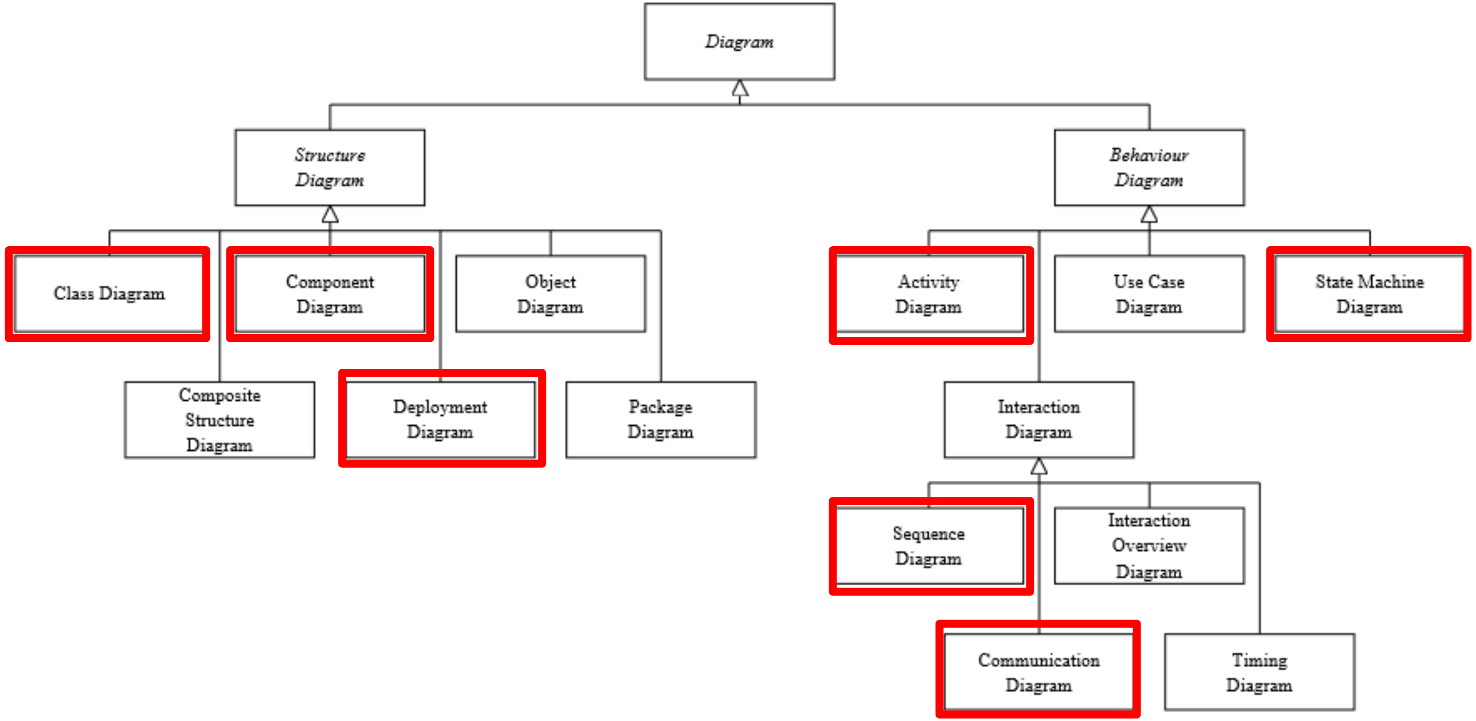


The Architecture Views
http://en.wikipedia.org/wiki/4%2B1_architectural_view_model

- ***Logical view*** : The logical view is concerned with the functionality that the system provides to end-users. UML Diagrams used to represent the logical view include Class diagram, Communication diagram, Sequence diagram.[2]
- ***Development view*** : The development view illustrates a system from a programmer's perspective and is concerned with software management. This view is also known as the implementation view. It uses the UML Component diagram to describe system components. UML Diagrams used to represent the development view include the Package diagram.[2]
- ***Process view*** : The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behavior of the system. The process view addresses concurrency, distribution, integrators, performance, and scalability, etc. UML Diagrams to represent process view include the Activity diagram.[2]
- ***Physical view*** : The physical view depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer, as well as the physical connections between these components. This view is also known as the deployment view. UML Diagrams used to represent physical view include the Deployment diagram.[2]
- ***Scenarios*** : The description of an architecture is illustrated using a small set of use cases, or scenarios which become a fifth view. The scenarios describe sequences of interactions between objects, and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. This view is **also known as use case view**

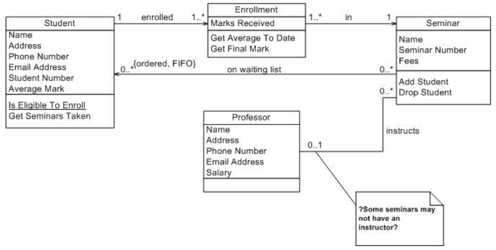
Some Diagrams

http://www.agilemodeling.com/essays/umlDiagrams.htm

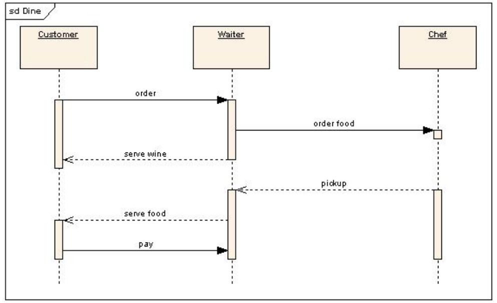


Some Diagrams

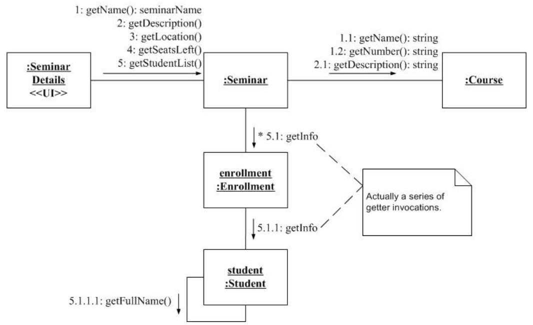
Class Diagram



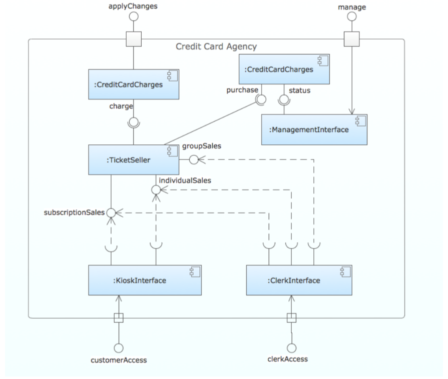
Sequence Diagram



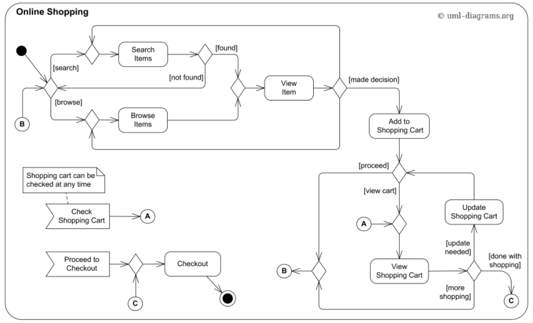
Communication Diagram



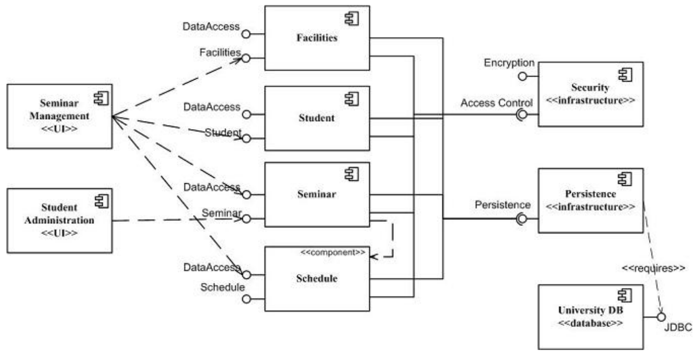
Component Diagram



Activity Diagram

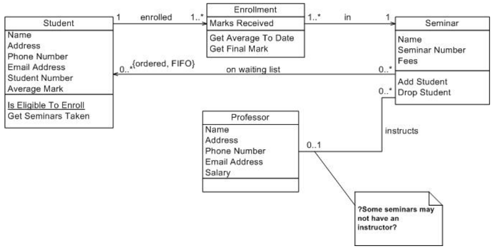


Component Diagram

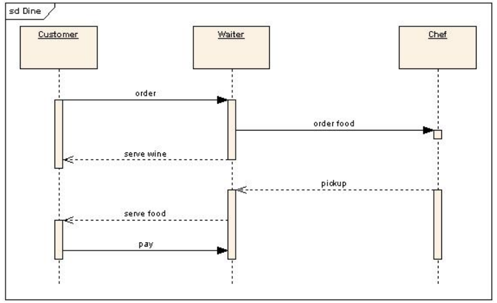


Application to 1st Project

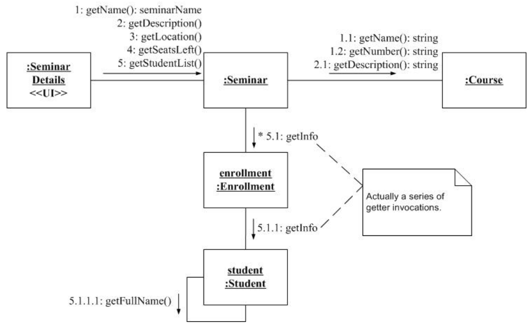
Class Diagram

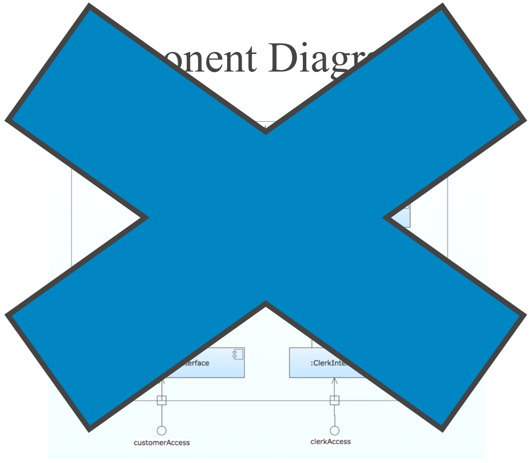


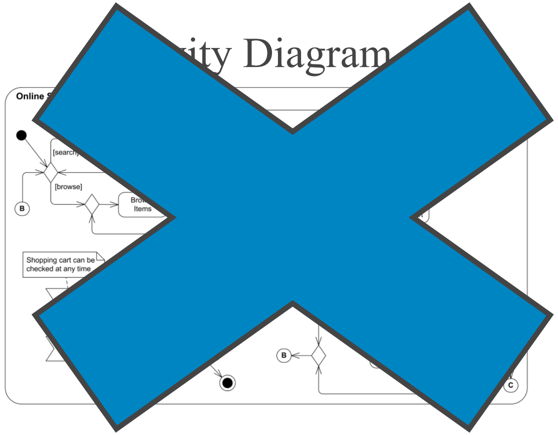
Sequence Diagram



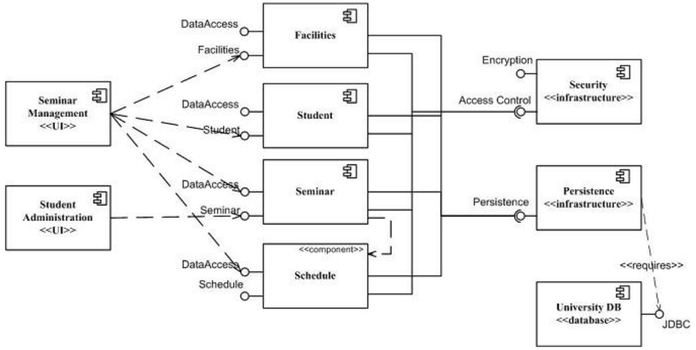
Communication Diagram







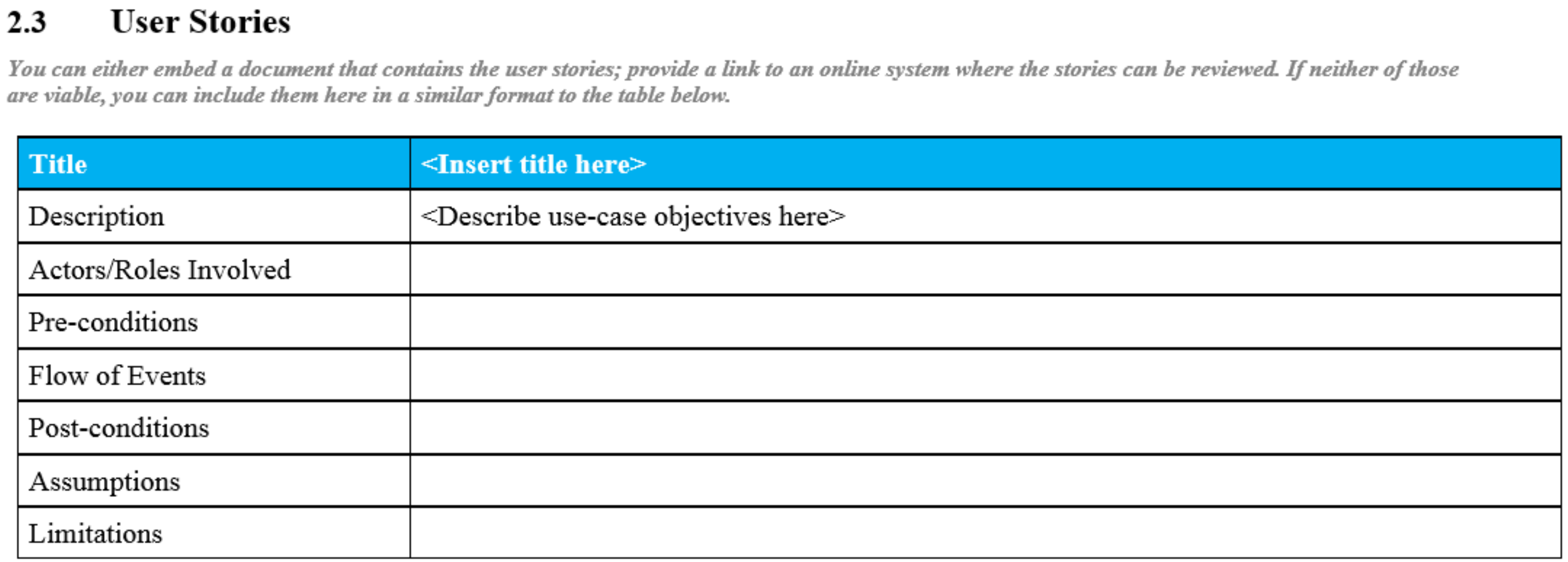
Component Diagram



User Stories

- **User stories** are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. They typically follow a simple template: (https://www.mountaingoatsoftware.com/agile/user-stories)
  - *As a <type of user>, I want <some goal> so that <some reason>.*
- Example user stories (http://www.agilemodeling.com/artifacts/userStory.htm)
  - Students can purchase monthly parking passes online.
  - Parking passes can be paid via credit cards.
  - Parking passes can be paid via PayPal.
  - Professors can input student marks.
  - Students can obtain their current seminar schedule.
  - Students can order official transcripts.
  - Students can only enroll in seminars for which they have prerequisites.
  - Transcripts will be available online via a standard browser

Agile Development and User Stories



2.3 User Stories

You can either embed a document that contains the user stories; provide a link to an online system where the stories can be reviewed. If neither of those are viable, you can include them here in a similar format to the table below.

Title	**<Insert title here>**
Description	<Describe use-case objectives here>
Actors/Roles Involved	
Pre-conditions	
Flow of Events	
Post-conditions	
Assumptions	
Limitations	

Documenting API Example

(We will use Swagger and Swagger Editor)

URI	https://mysite.com:3911/api/members/{id}
HTTP verb	**PUT**
Parameters	**id** : Card number of the member.
Body	**name** : Name of the member. **email** : Email adress of the member. **langage** : Langage used by member (Fr_CA ou En_US)

Sample body	{
	"name":"Mario Cardinal",
	"email":"mcardinal@mariocardinal.com",
	"language":"fr_CA"
	}

Success Response	Status Code: 204 No Content
Error Response	**Status Code**: 400 Bad Request, **Body**: {"Error Code":"..."} **Status Code**: 401 Unauthenticated, see **WWW-Authenticate** value in header **Status Code**: 403 Forbidden **Status Code**: 404 Not Found **Status Code**: 429 Too Many Requests, see **Retry-After** value in header **Status Code**: 500 Internal Server Error **Status Code**: 503 Service Unavailable
Error Code	**10**: Inactive member **20**: Denied access member **110**: Database issues, Retry later

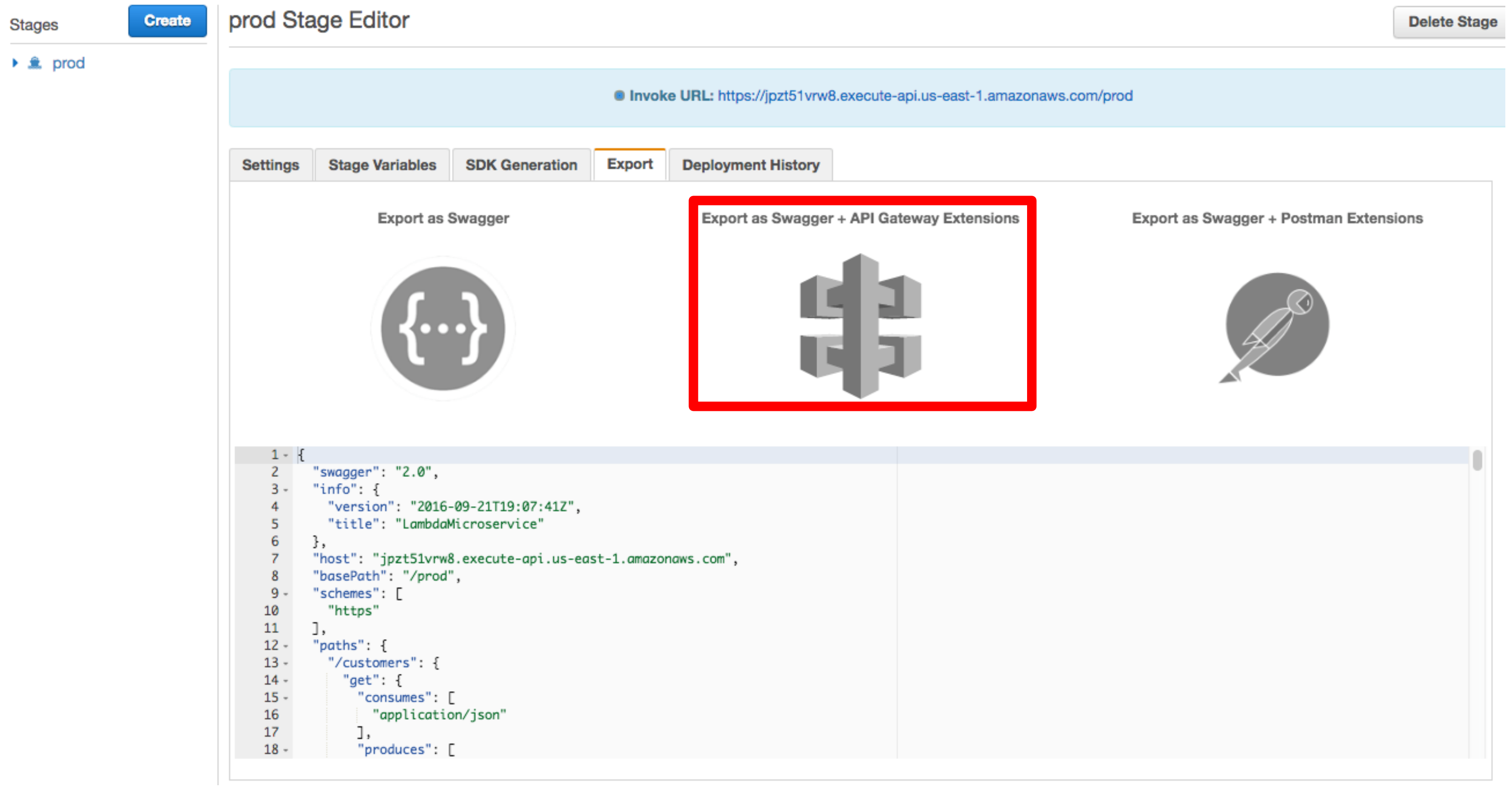
Top-Level Design Specification

- Comments
  - I have presented a very *systematic* version of TLDS content.
  - All development projects should do *some flavor* of this approach, ideally simple and living/dynamic.
  - Just want to give you a feel for the types of things you can include in your TLDS, but you only need a really simple version.
  - Use common sense.
- For your TLDS
  - You do not need to use UML, but should use some of the concepts.
  - Some useful diagrams
    - Component diagram.
    - Class diagram documenting the logical datamodels/ resource models for components.
    - Some sequence diagrams showing end-to-end flows and Lambda functions in flows.
- Define and document
  - A couple of user stories.
  - Your APIs.

New Topics

Swagger

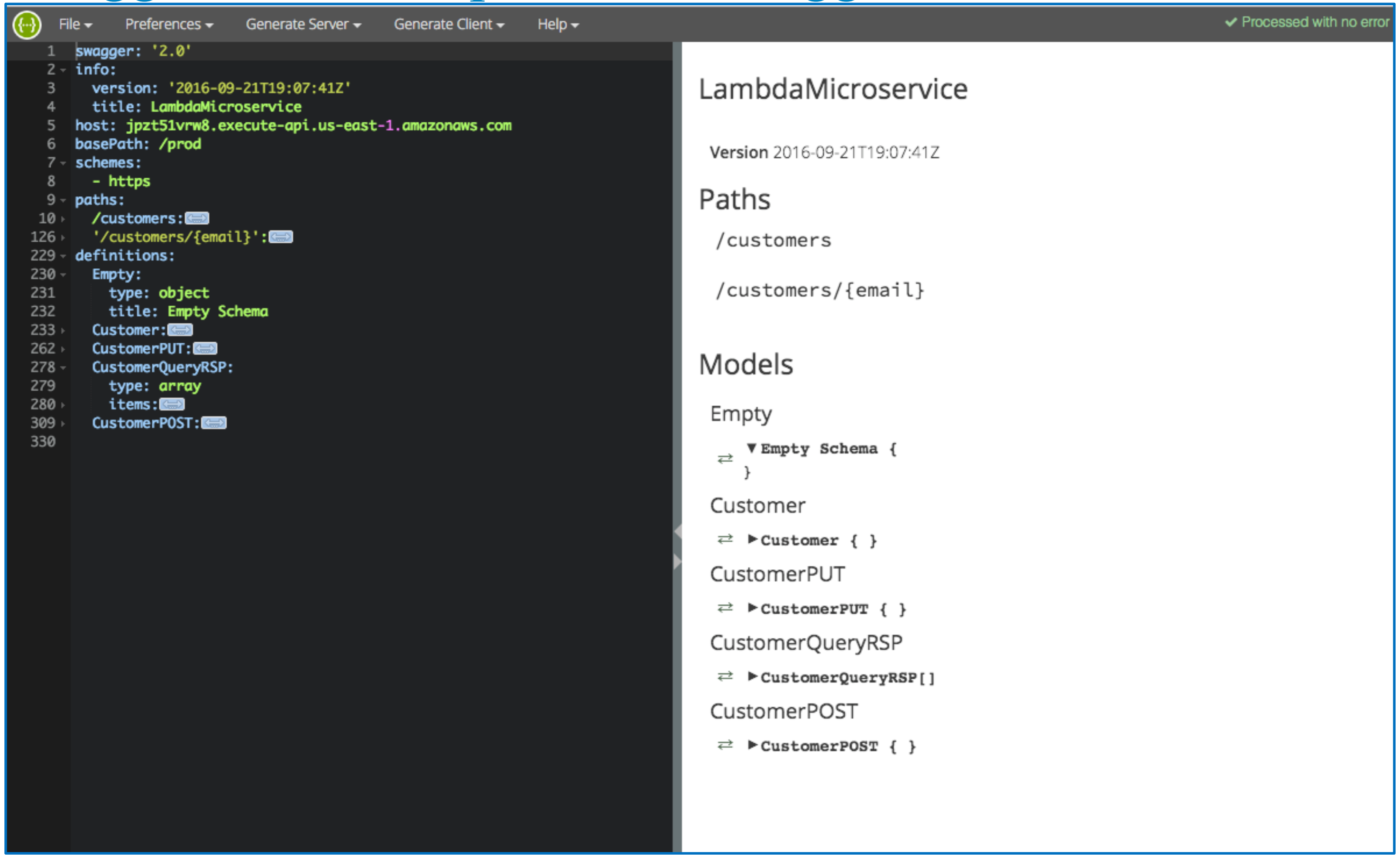
Swagger



Invoke URL: https://jpzt51vrw8.execute-api.us-east-1.amazonaws.com/prod

```
{
  "swagger": "2.0",
  "info": {
    "version": "2016-09-21T19:07:41Z",
    "title": "LambdaMicroservice"
  },
  "host": "jpzt51vrw8.execute-api.us-east-1.amazonaws.com",
  "basePath": "/prod",
  "schemes": [
    "https"
  ],
  "paths": {
    "/customers": {
      "get": {
        "consumes": [
          "application/json"
        ],
        "produces": [
```

Swagger Editor: http://editor.swagger.io/#/



```
swagger: '2.0'
info:
  version: '2016-09-21T19:07:41Z'
  title: LambdaMicroservice
host: jpzt51vrw8.execute-api.us-east-1.amazonaws.com
basePath: /prod
schemes:
  - https
paths:
  /customers:
  '/customers/{email}':
definitions:
  Empty:
    type: object
    title: Empty Schema
  Customer:
  CustomerPUT:
  CustomerQueryRSP:
    type: array
    items:
  CustomerPOST:
```

LambdaMicroservice

Version 2016-09-21T19:07:41Z

Paths

/customers

/customers/{email}

Models

Empty

▼Empty Schema {
}

Customer

▶Customer { }

CustomerPUT

▶CustomerPUT { }

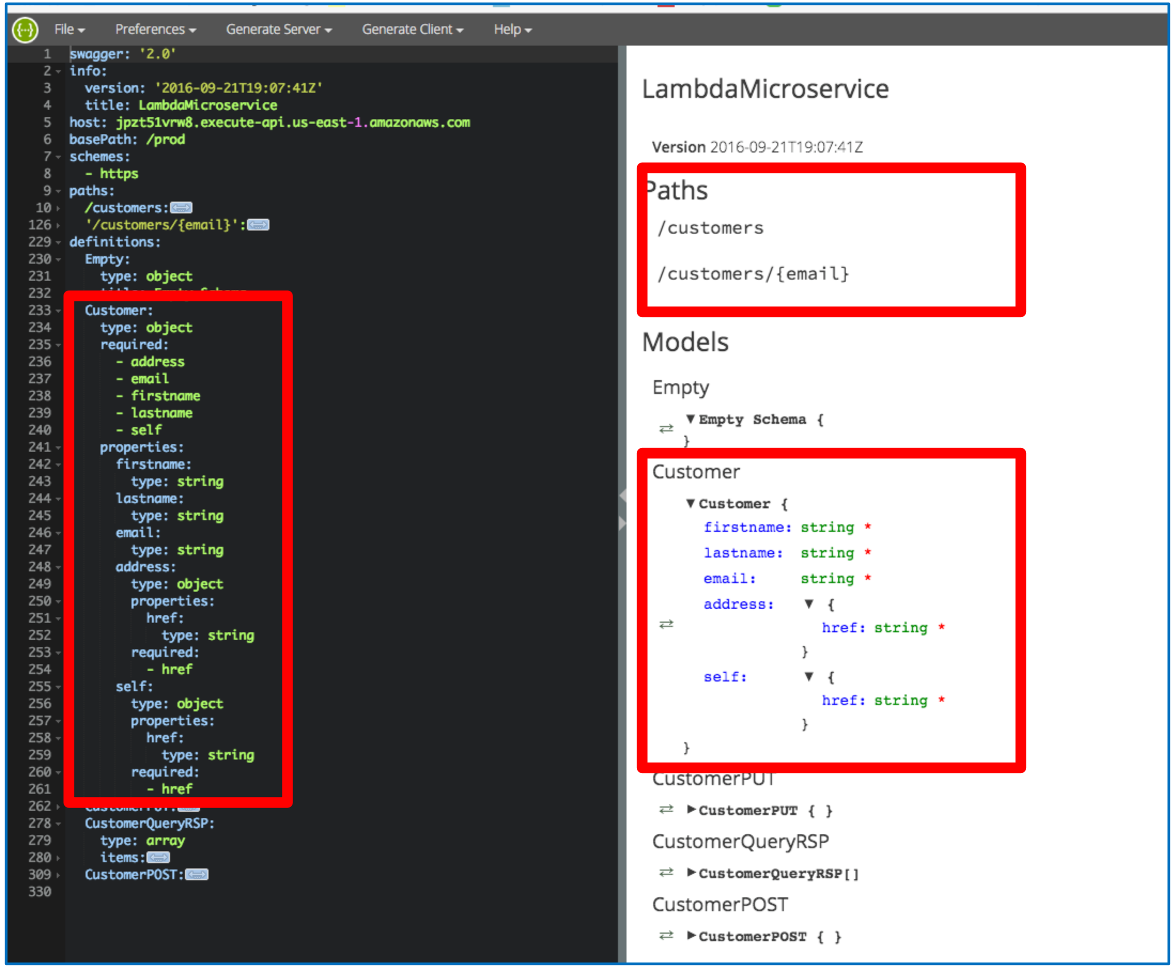
CustomerQueryRSP

▶CustomerQueryRSP[]

CustomerPOST

▶CustomerPOST { }

Models



```
swagger: '2.0'
info:
  version: '2016-09-21T19:07:41Z'
  title: LambdaMicroservice
host: jpzt51vrw8.execute-api.us-east-1.amazonaws.com
basePath: /prod
schemes:
  - https
paths:
  /customers:
  '/customers/{email}':
definitions:
  Empty:
    type: object
    title: Empty Schema
  Customer:
    type: object
    required:
      - address
      - email
      - firstname
      - lastname
      - self
    properties:
      firstname:
        type: string
      lastname:
        type: string
      email:
        type: string
      address:
        type: object
        properties:
          href:
            type: string
        required:
          - href
      self:
        type: object
        properties:
          href:
            type: string
        required:
          - href
  CustomerPUT:
  CustomerQueryRSP:
    type: array
    items:
  CustomerPOST:
```

LambdaMicroservice

Version 2016-09-21T19:07:41Z

Paths

/customers

/customers/{email}

Models

Empty

```
▼Empty Schema {
}
```

Customer

```
▼Customer {
   firstname:  string *
   lastname:   string *
   email:      string *
   address:    ▼ {
                  href: string *
               }
   self:       ▼ {
                  href: string *
               }
}
```

CustomerPUT

```
▶CustomerPUT { }
```

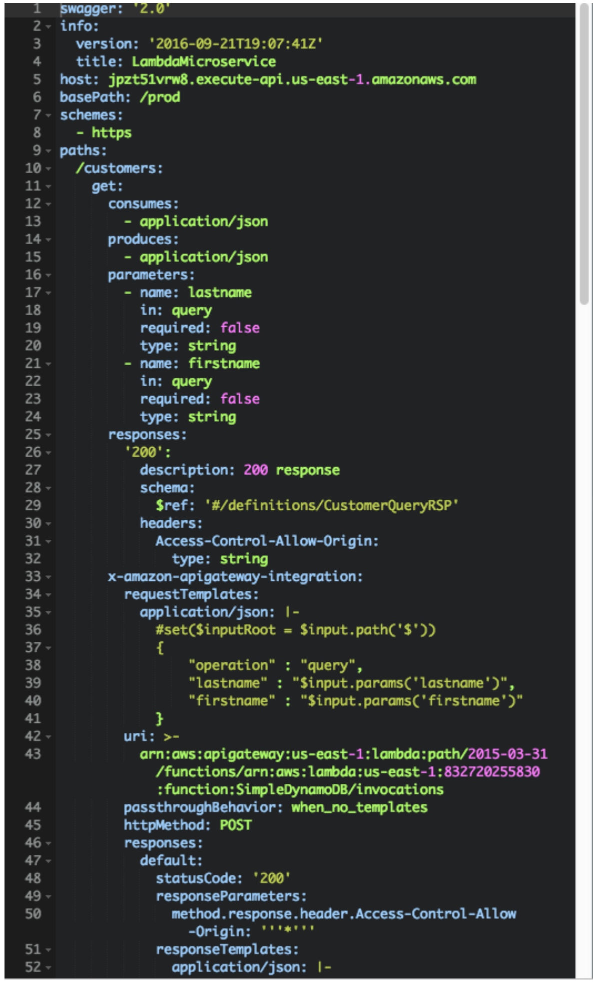
CustomerQueryRSP

```
▶CustomerQueryRSP[]
```

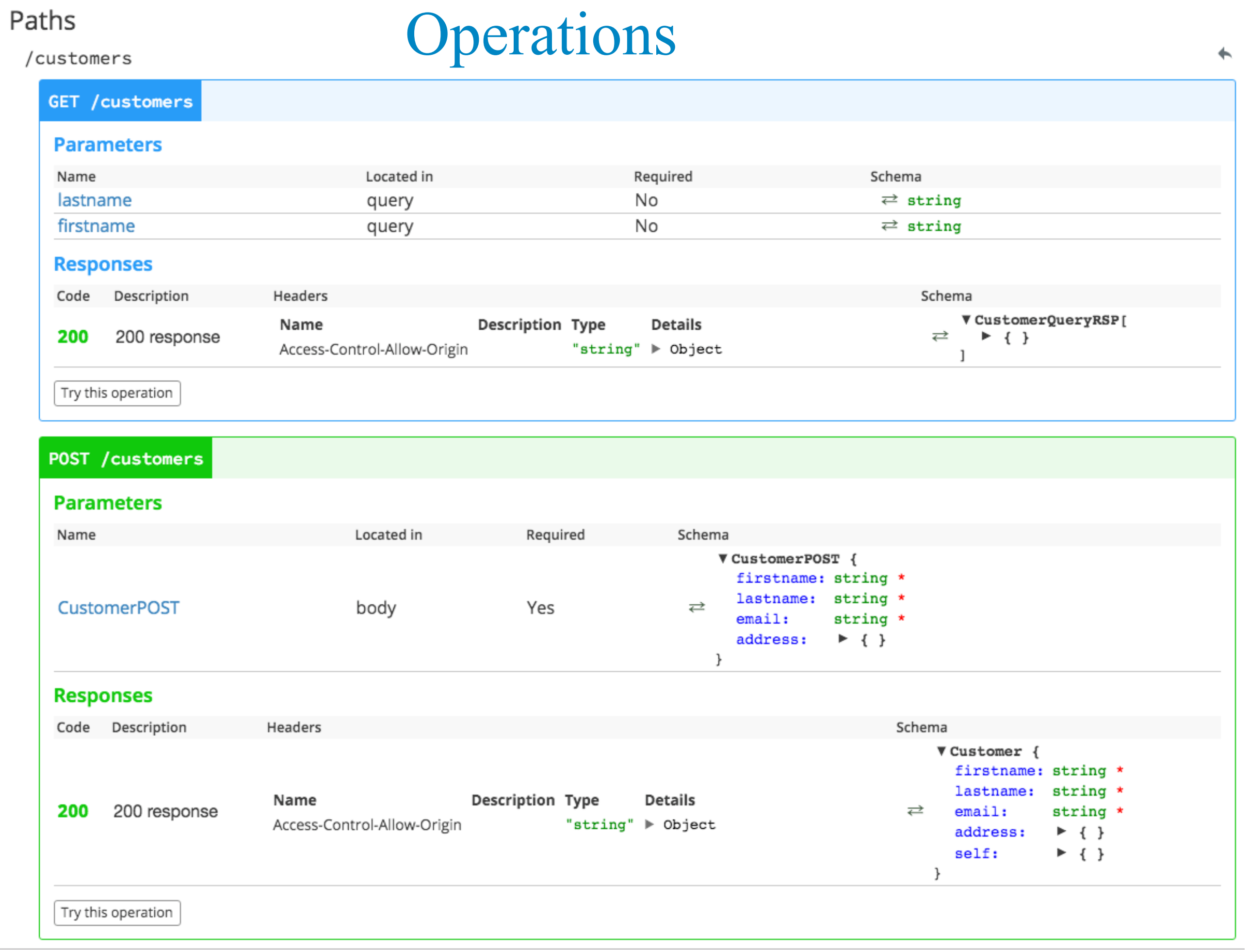
CustomerPOST

```
▶CustomerPOST { }
```

Operations



```
swagger: '2.0'
info:
  version: '2016-09-21T19:07:41Z'
  title: LambdaMicroservice
host: jpzt51vrw8.execute-api.us-east-1.amazonaws.com
basePath: /prod
schemes:
  - https
paths:
  /customers:
    get:
      consumes:
        - application/json
      produces:
        - application/json
      parameters:
        - name: lastname
          in: query
          required: false
          type: string
        - name: firstname
          in: query
          required: false
          type: string
      responses:
        '200':
          description: 200 response
          schema:
            $ref: '#/definitions/CustomerQueryRSP'
          headers:
            Access-Control-Allow-Origin:
              type: string
      x-amazon-apigateway-integration:
        requestTemplates:
          application/json: |-
            #set($inputRoot = $input.path('$'))
            {
                "operation" : "query",
                "lastname" : "$input.params('lastname')",
                "firstname" : "$input.params('firstname')"
            }
        uri: >-
          arn:aws:apigateway:us-east-1:lambda:path/2015-03-31
            /functions/arn:aws:lambda:us-east-1:832720255830
            :function:SimpleDynamoDB/invocations
        passthroughBehavior: when_no_templates
        httpMethod: POST
        responses:
          default:
            statusCode: '200'
            responseParameters:
              method.response.header.Access-Control-Allow
                -Origin: '''*'''
            responseTemplates:
              application/json: |-
```



Paths

/customers

GET /customers

Parameters

Name	Located in	Required	Schema
lastname	query	No	⇄ string
firstname	query	No	⇄ string

Responses

Code	Description	Headers	Schema
200	200 response	Name: Access-Control-Allow-Origin; Description: ; Type: "string"; Details: ▶ Object	⇄ ▼CustomerQueryRSP[▶ { }]

Try this operation

POST /customers

Parameters

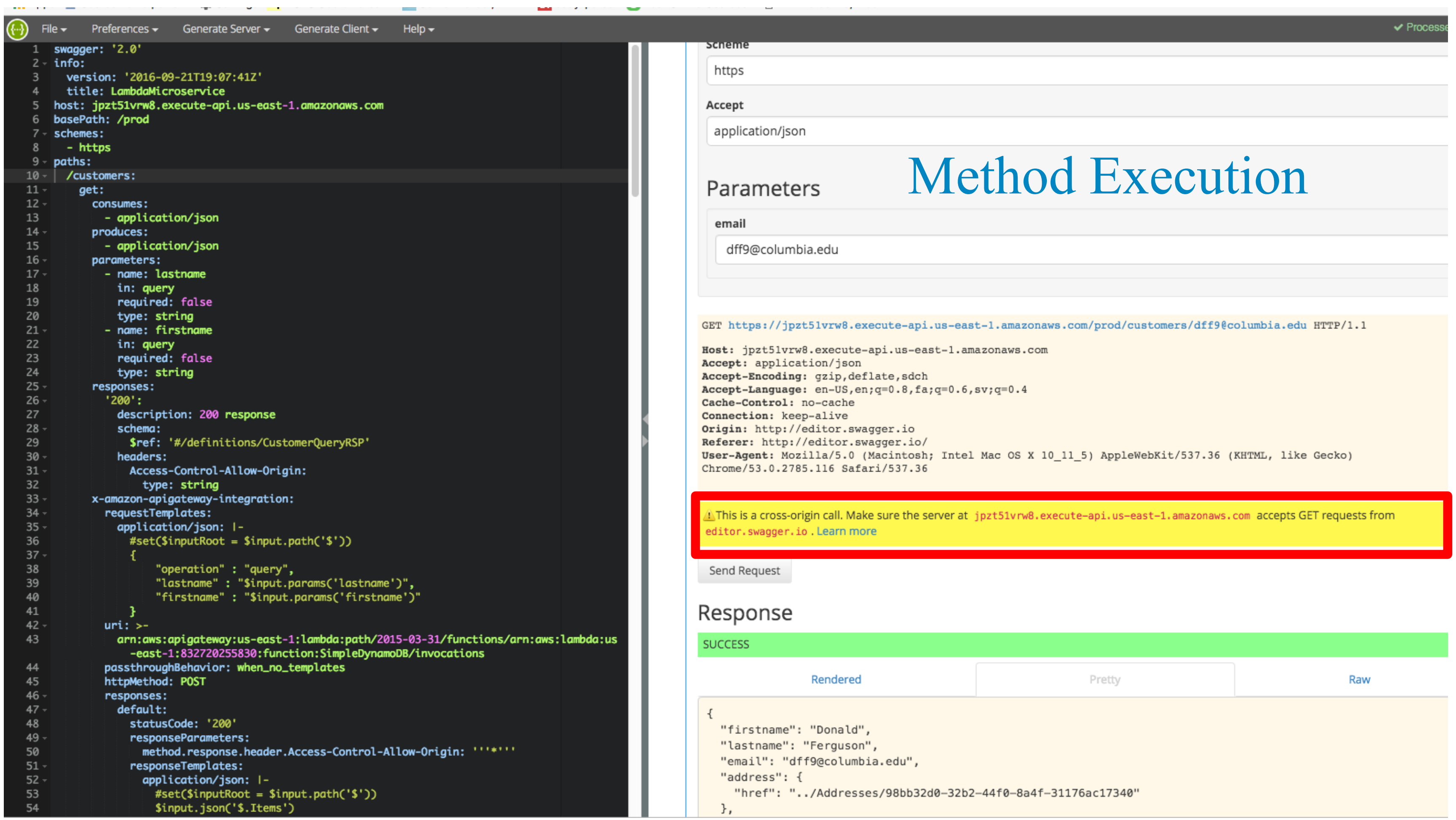
Name	Located in	Required	Schema
CustomerPOST	body	Yes	⇄ ▼CustomerPOST { firstname: string *, lastname: string *, email: string *, address: ▶ { } }

Responses

Code	Description	Headers	Schema
200	200 response	Name: Access-Control-Allow-Origin; Description: ; Type: "string"; Details: ▶ Object	⇄ ▼Customer { firstname: string *, lastname: string *, email: string *, address: ▶ { }, self: ▶ { } }

Try this operation

Method Execution



```
swagger: '2.0'
info:
  version: '2016-09-21T19:07:41Z'
  title: LambdaMicroservice
host: jpzt51vrw8.execute-api.us-east-1.amazonaws.com
basePath: /prod
schemes:
  - https
paths:
  /customers:
    get:
      consumes:
        - application/json
      produces:
        - application/json
      parameters:
        - name: lastname
          in: query
          required: false
          type: string
        - name: firstname
          in: query
          required: false
          type: string
      responses:
        '200':
          description: 200 response
          schema:
            $ref: '#/definitions/CustomerQueryRSP'
          headers:
            Access-Control-Allow-Origin:
              type: string
      x-amazon-apigateway-integration:
        requestTemplates:
          application/json: |-
            #set($inputRoot = $input.path('$'))
            {
                "operation" : "query",
                "lastname" : "$input.params('lastname')",
                "firstname" : "$input.params('firstname')"
            }
        uri: >-
          arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us
            -east-1:832720255830:function:SimpleDynamoDB/invocations
        passthroughBehavior: when_no_templates
        httpMethod: POST
        responses:
          default:
            statusCode: '200'
            responseParameters:
              method.response.header.Access-Control-Allow-Origin: '''*'''
            responseTemplates:
              application/json: |-
                #set($inputRoot = $input.path('$'))
                $input.json('$.Items')
```

Scheme

https

Accept

application/json

Parameters

email

dff9@columbia.edu

```
GET https://jpzt51vrw8.execute-api.us-east-1.amazonaws.com/prod/customers/dff9@columbia.edu HTTP/1.1

Host: jpzt51vrw8.execute-api.us-east-1.amazonaws.com
Accept: application/json
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8,fa;q=0.6,sv;q=0.4
Cache-Control: no-cache
Connection: keep-alive
Origin: http://editor.swagger.io
Referer: http://editor.swagger.io/
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_5) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/53.0.2785.116 Safari/537.36
```

⚠This is a cross-origin call. Make sure the server at jpzt51vrw8.execute-api.us-east-1.amazonaws.com accepts GET requests from editor.swagger.io . Learn more

Send Request

Response

SUCCESS

Rendered | Pretty | Raw

```
{
  "firstname": "Donald",
  "lastname": "Ferguson",
  "email": "dff9@columbia.edu",
  "address": {
    "href": "../Addresses/98bb32d0-32b2-44f0-8a4f-31176ac17340"
  },
```

http://www.slideshare.net/VictorTrakhtenberg/swagger2

Swagger is…

Technology

A framework

for

- producing
- consuming
- visualizing

RESTful APIs

Methodology

A specification

for

- describing
- documenting

RESTful APIs

Getting Started

http://swagger.io/getting-started/

Introduction to Swagger

Tony Tam
@fehguy

Why Swagger?

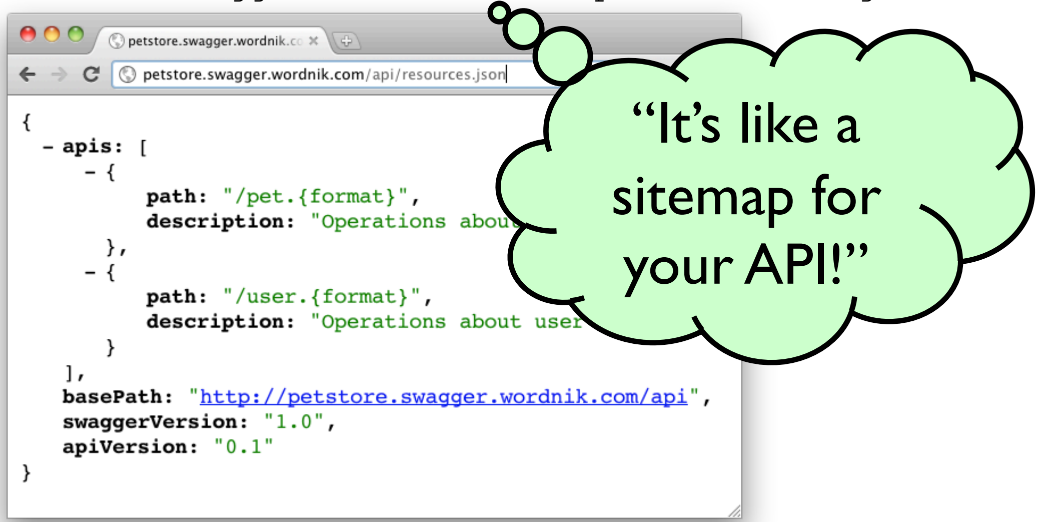
- Integration with REST APIs is troublesome and inconsistent
  - Different vendors have different REST semantics
  - Client libraries vary wildly by vendor, language
  - Documentation for developers is an afterthought
  - Input parameters, allowable values, models, responses are found via trial & error
- Internally a PITA
- YOUR API is too hard to develop against!

wordnik

How Does it Work?

- Your server produces a Resource List
  - All available APIs

`http://petstore.swagger.wordnik.com/api/resources.json`


petstore.swagger.wordnik.co
petstore.swagger.wordnik.com/api/resources.json
"It's like a sitemap for your API!"

```
{
  - apis: [
      - {
            path: "/pet.{format}",
            description: "Operations abou
        },
      - {
            path: "/user.{format}",
            description: "Operations about user
        }
    ],
    basePath: "http://petstore.swagger.wordnik.com/api",
    swaggerVersion: "1.0",
    apiVersion: "0.1"
}
```

wordnik

How Does it Work?

- Each API declares itself
  - Available operations
  - Parameters
    - Type (path, query, body)
    - Allowable values/data types
  - Input/output models
  - Error responses with descriptions

Client Library Generation

- Code generation based on Resource Spec
  - Template-based Framework
    - Consume REST Methods, Models, Parameters
    - Produce client libraries
- “Know before you go HTTP”
  - Required values, fields are known by the client
- Only expose what’s allowed!
  - Swagger filtering removes methods/models you don’t have access to

Client Library Generation

- Code generatio ased on Resource Spec
  - Template-based work
    - Consume RES ds, Models, Parameters
    - Produce client
- "Know before yo "
  - Require
- Only exp
  - Swagge to

Also generation for

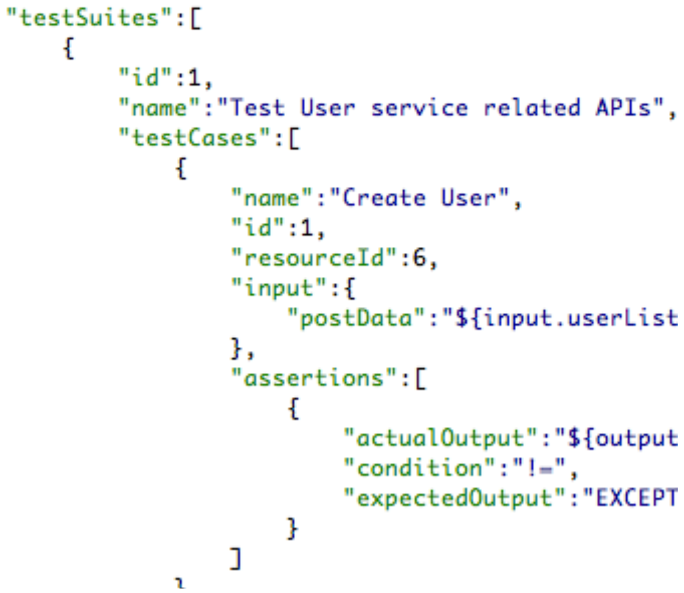
- Servers, although limited for Lambda.
- API Gateway.
- … ...

wordnik

Test Framework

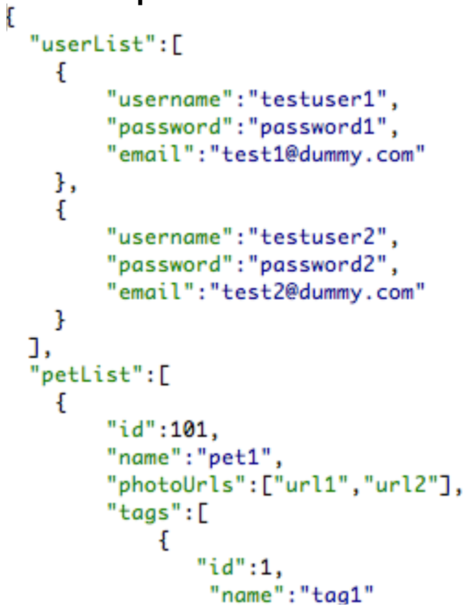
- JSON-driven tests for your Client + API

Test Suites



```
"testSuites":[
    {
        "id":1,
        "name":"Test User service related APIs",
        "testCases":[
            {
                "name":"Create User",
                "id":1,
                "resourceId":6,
                "input":{
                    "postData":"${input.userList
                },
                "assertions":[
                    {
                        "actualOutput":"${output
                        "condition":"!=",
                        "expectedOutput":"EXCEPT
                    }
                ]
```

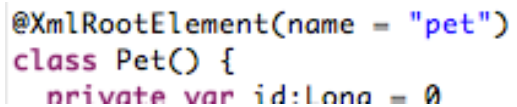
Expected data



```
{
  "userList":[
    {
        "username":"testuser1",
        "password":"password1",
        "email":"test1@dummy.com"
    },
    {
        "username":"testuser2",
        "password":"password2",
        "email":"test2@dummy.com"
    }
  ],
  "petList":[
    {
        "id":101,
        "name":"pet1",
        "photoUrls":["url1","url2"],
        "tags":[
            {
                "id":1,
                "name":"tag1"
```

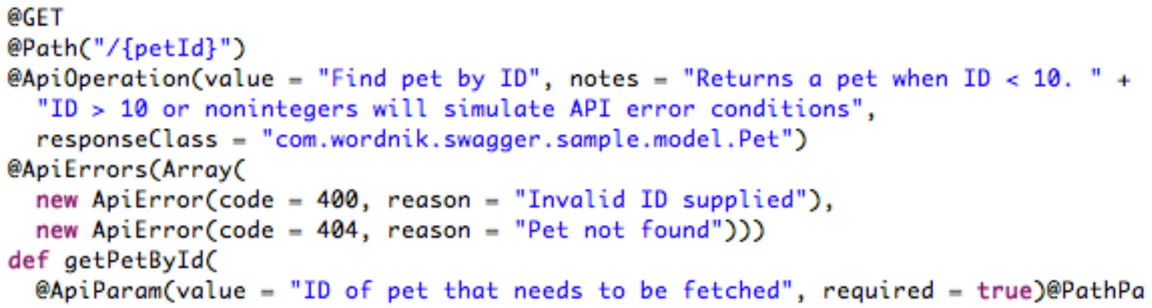
Easy to add

- For Java/Scala via JAX-RS…
  - Add swagger-core.jar
  - Annotate your models per your `@Provider` class



```
@XmlRootElement(name = "pet")
class Pet() {
  private var id:Long = 0
```

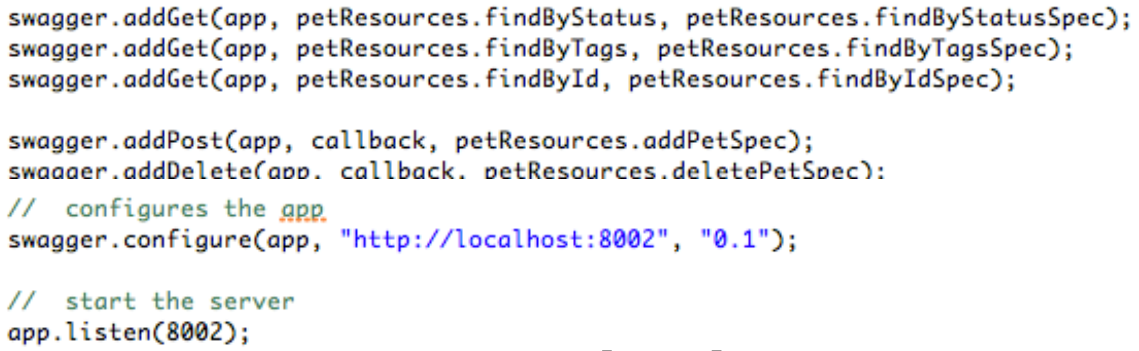
  - Annotate your resources



```
@GET
@Path("/{petId}")
@ApiOperation(value = "Find pet by ID", notes = "Returns a pet when ID < 10. " +
  "ID > 10 or nonintegers will simulate API error conditions",
  responseClass = "com.wordnik.swagger.sample.model.Pet")
@ApiErrors(Array(
  new ApiError(code = 400, reason = "Invalid ID supplied"),
  new ApiError(code = 404, reason = "Pet not found")))
def getPetById(
  @ApiParam(value = "ID of pet that needs to be fetched", required = true)@PathPa
```

Easy to add

- For node.js via express
  - Require swagger.js
  - Declare your swagger specs, models
    - Could use AST/DSL to do automatically
  - Add your operations, configure and start



```
swagger.addGet(app, petResources.findByStatus, petResources.findByStatusSpec);
swagger.addGet(app, petResources.findByTags, petResources.findByTagsSpec);
swagger.addGet(app, petResources.findById, petResources.findByIdSpec);

swagger.addPost(app, callback, petResources.addPetSpec);
swagger.addDelete(app, callback, petResources.deletePetSpec);
//  configures the app
swagger.configure(app, "http://localhost:8002", "0.1");

//  start the server
app.listen(8002);
```

Easy DIY

- The Swagger spec is Language Agnostic!
    - ANY swagger-compliant server can…
        - Use the Swagger client lib generator
        - Use the test framework
        - Use the sandbox UI
- More server support from Wordnik
    - Play, Rails
- .net, others community developed

wordnik

- Methodology for
  - Designing
  - Building
  - Testing
  - Finding/Using

“Services”

- Serverless/AWS is
  - Programming Model
  - For Implementing *methods* with
    - Agility
    - Flexibility
    - Efficiency
    - … …

Definition Editor

Use our intuitive editors to create your API definition and collaborate with others on their API definitions. On-the-fly validation keeps you honest.

LEARN MORE

Code Gen

Get your development project off to a quick start by using our client and server code templates.

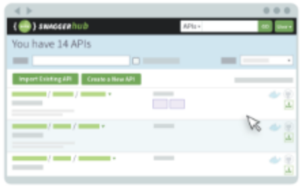
LEARN MORE

Versioning

Manage different versions of your API definition and decide for yourself when to publish a version or push it to GitHub.

LEARN MORE

API Registry



Browse our list of Swagger-based APIs and explore them using our interactive documentation.

LEARN MORE

Domains

Store all your re-usable components that can later be used across multiple API definitions, saving you time and maximizing productivity

LEARN MORE

Plugins

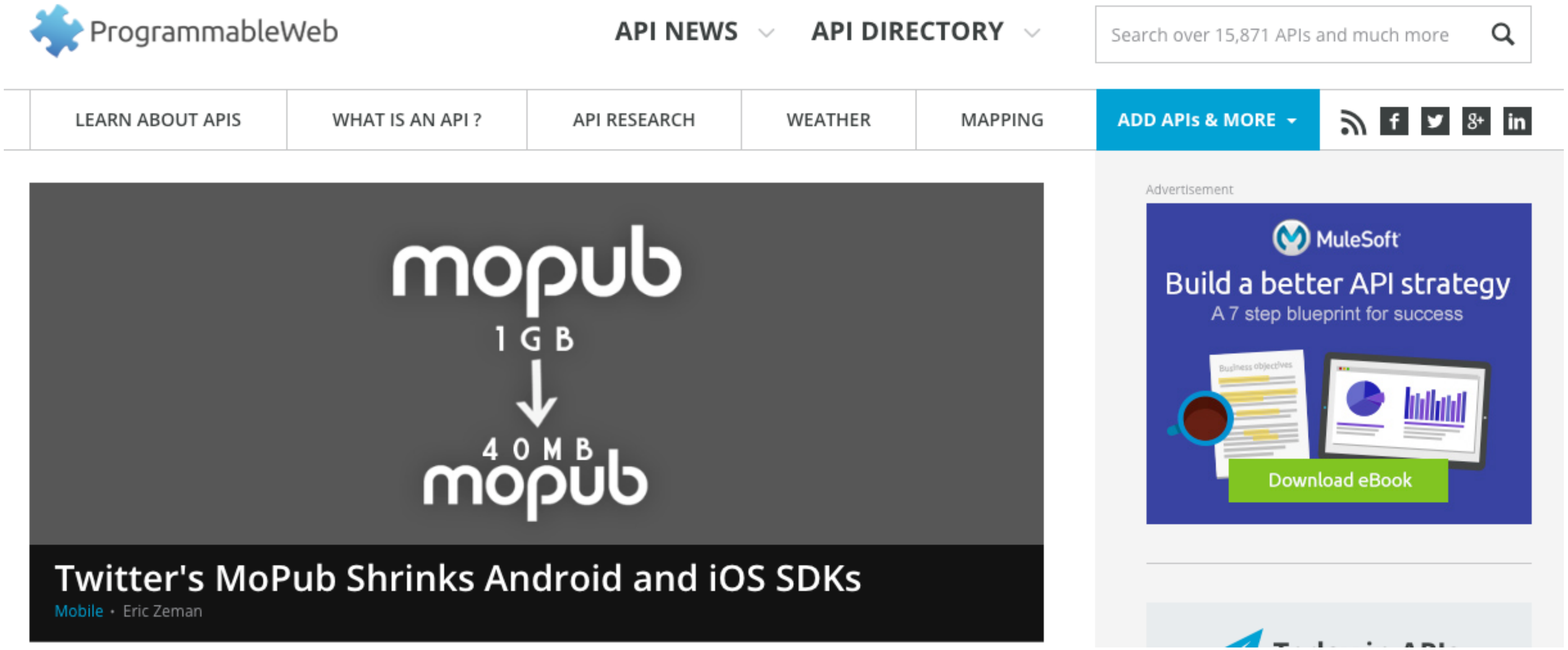
LEARN MORE

Let's Walk Through the Example

http://swagger.io/getting-started/

Programmable Web

www.programmableweb.com


ProgrammableWeb

API NEWS

API DIRECTORY

Search over 15,871 APIs and much more

LEARN ABOUT APIS

WHAT IS AN API ?

API RESEARCH

WEATHER

MAPPING

ADD APIs & MORE

Twitter's MoPub Shrinks Android and iOS SDKs

Mobile • Eric Zeman

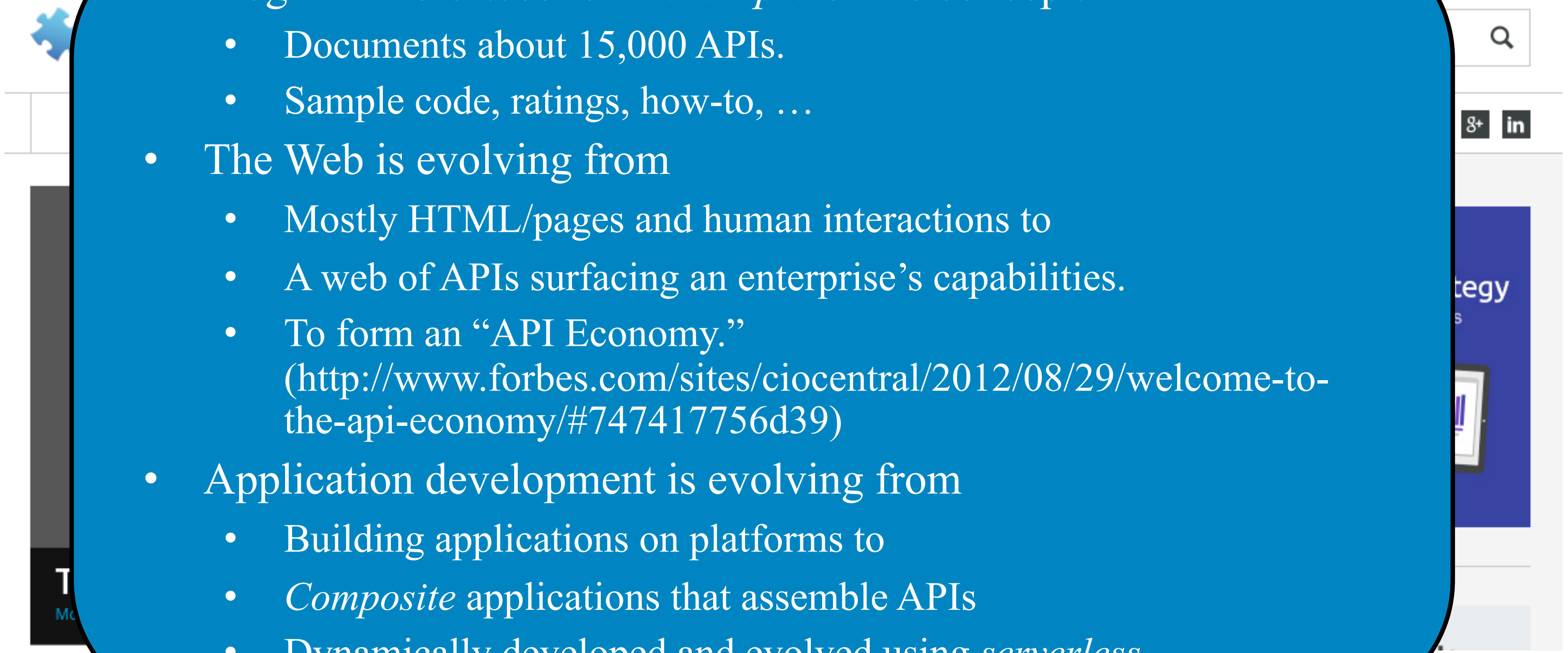
Advertisement

MuleSoft

Build a better API strategy

A 7 step blueprint for success

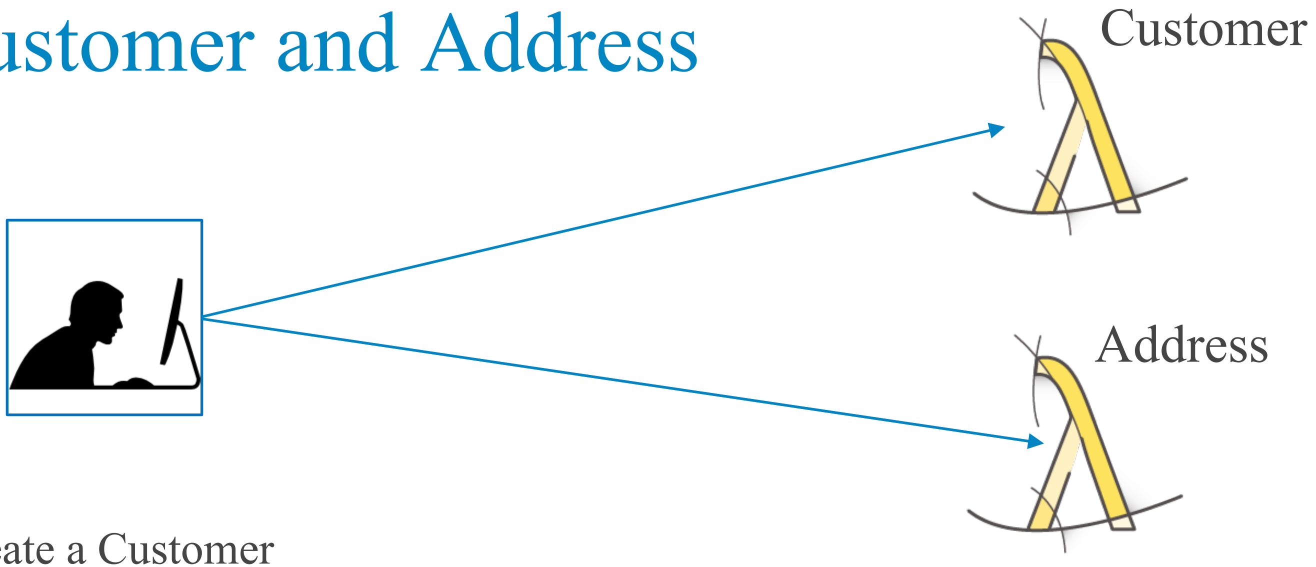
Download eBook



- Programmable Web is an *example* of the concept.
  - Documents about 15,000 APIs.
  - Sample code, ratings, how-to, …
- The Web is evolving from
  - Mostly HTML/pages and human interactions to
  - A web of APIs surfacing an enterprise’s capabilities.
  - To form an “API Economy.” (http://www.forbes.com/sites/ciocentral/2012/08/29/welcome-to-the-api-economy/#747417756d39)
- Application development is evolving from
  - Building applications on platforms to
  - *Composite* applications that assemble APIs
  - Dynamically developed and evolved using *serverless*.

2nd Project

Customer and Address



Create a Customer

- I can enter the address information and customer info on one "form" and make two calls.
- How do I know if
  - The address does/does not already exist?
  - Do I completely match the text? What if one used "S. Salem" and one used "South Salem?"
  - If the address already exists, how do I give the user the option of choosing without showing all of the address in a zipcode?
  - … ...

Reuse an API

Why write a service

And incrementally

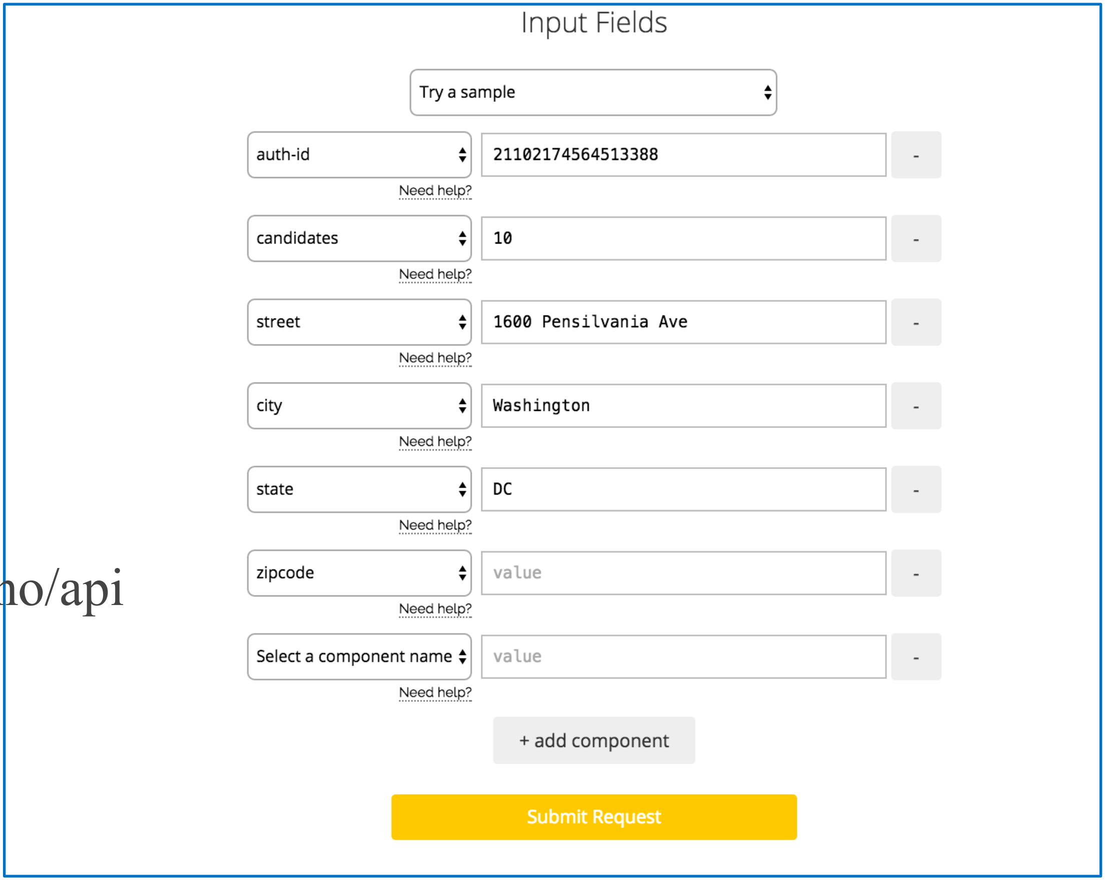
Build data if

Someone has

Already done it?

For example

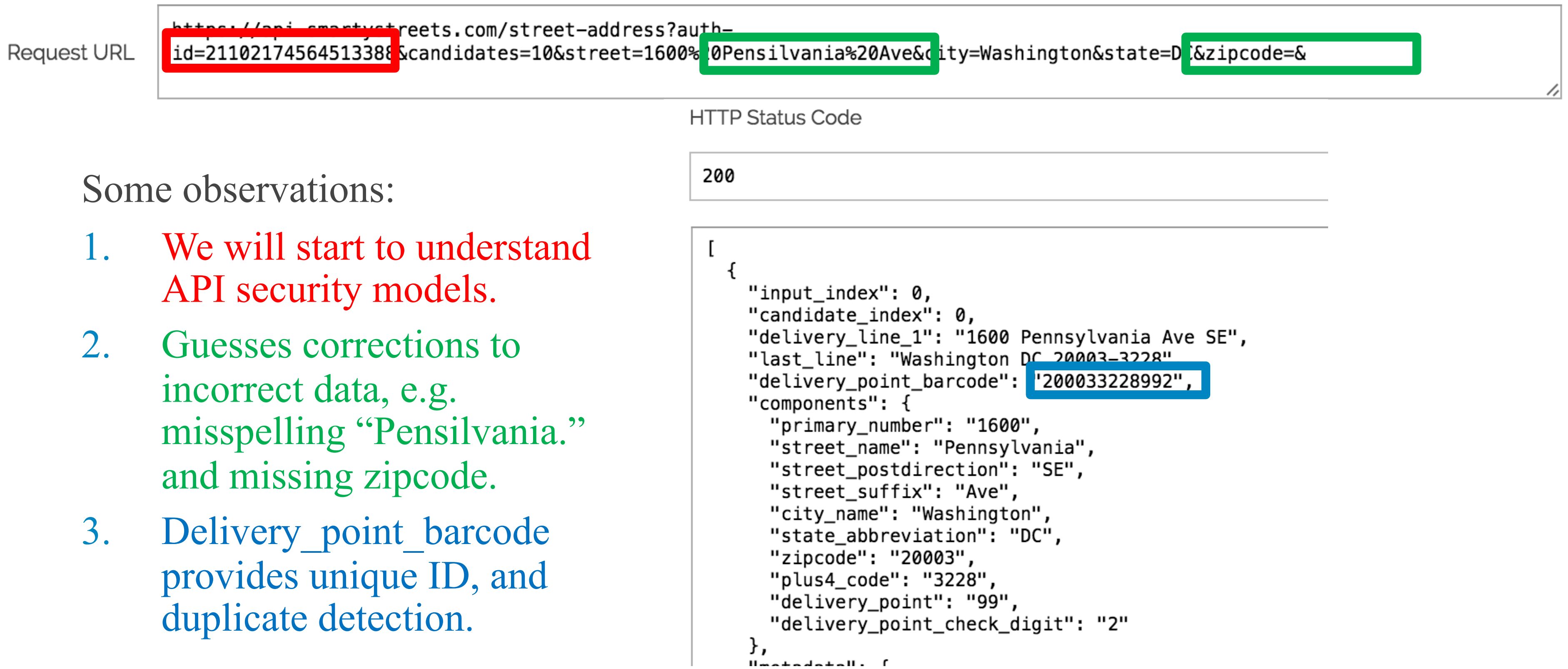
https://smartystreets.com/demo/api


Input Fields

Try a sample

auth-id	21102174564513388
candidates	10
street	1600 Pensilvania Ave
city	Washington
state	DC
zipcode	value
Select a component name	value

Example REST Call and Response



Request URL

```
https://api.smartystreets.com/street-address?auth-
id=21102174564513388&candidates=10&street=1600%20Pensilvania%20Ave&city=Washington&state=DC&zipcode=&
```

HTTP Status Code

```
200
```

```
[
  {
    "input_index": 0,
    "candidate_index": 0,
    "delivery_line_1": "1600 Pennsylvania Ave SE",
    "last_line": "Washington DC 20003-3228",
    "delivery_point_barcode": "200033228992",
    "components": {
      "primary_number": "1600",
      "street_name": "Pennsylvania",
      "street_postdirection": "SE",
      "street_suffix": "Ave",
      "city_name": "Washington",
      "state_abbreviation": "DC",
      "zipcode": "20003",
      "plus4_code": "3228",
      "delivery_point": "99",
      "delivery_point_check_digit": "2"
    },
    "metadata": {
```

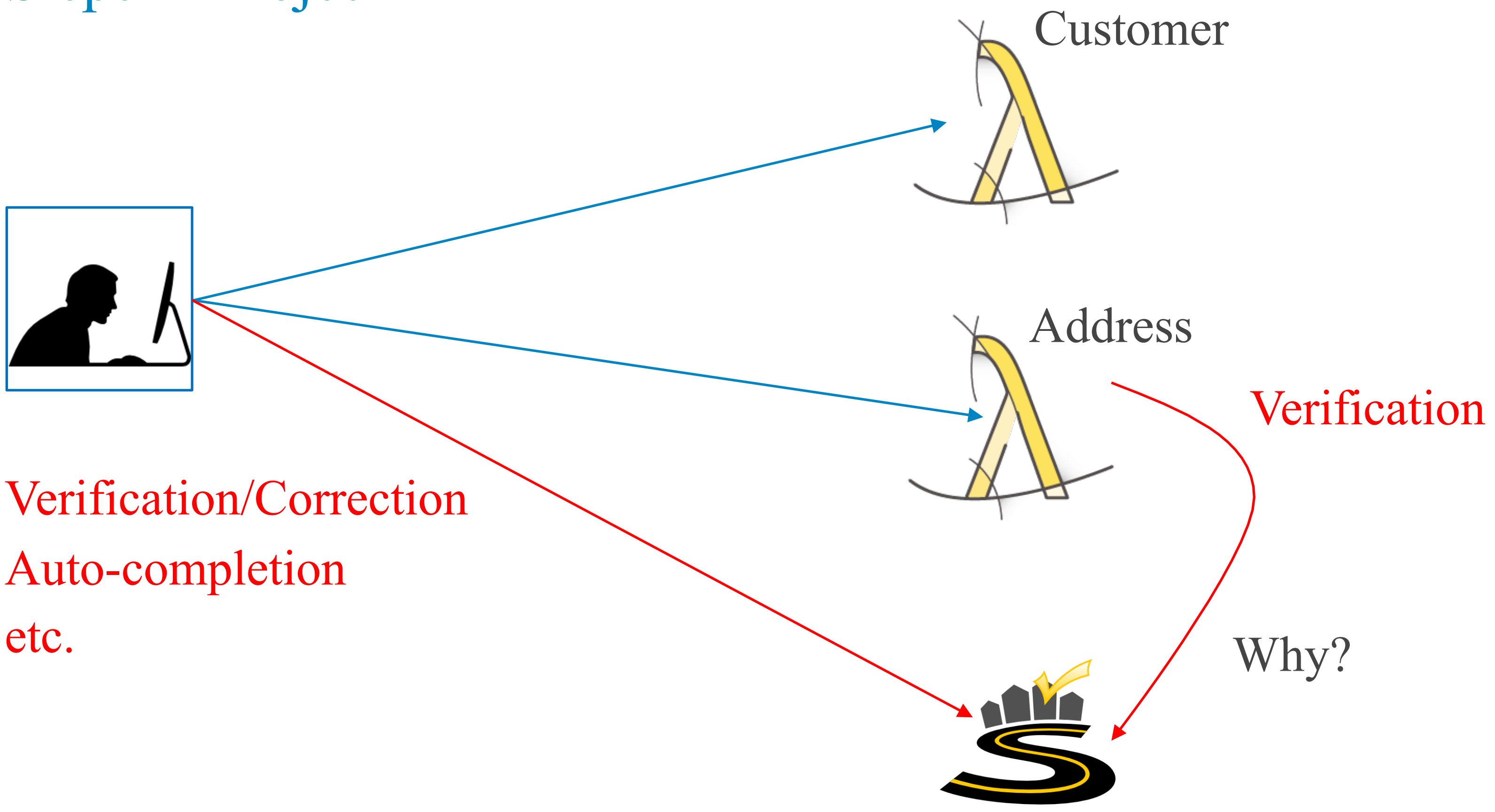
Some observations:

1. We will start to understand API security models.
2. Guesses corrections to incorrect data, e.g. misspelling "Pensilvania." and missing zipcode.
3. Delivery_point_barcode provides unique ID, and duplicate detection.

Auto-Complete Demo

https://smartystreets.com/demo

Next Steps in Project



- Verification/Correction
- Auto-completion
- etc.

Assignment 2

- Expand on Assignment #1.
- Expand API functionality
  - Fully implement success and error codes.
  - Href for address link and self.
  - Full Swagger definition, including models.
- Build a simple, ideally AngularJS UI
  - CRUD customer information forms.
  - Forms to add and change the address associated with a customer, including auto-complete. (www.smartystreets.com free tier).
- Modify Dynamo DB and Lambda function(s)
  - Replace address UUID with Delivery_Point_Barcode.
  - An address appears once (is unique) in the DynamoDB table.
  - Call www.smartystreets.com to verify on PUT/POST, and throw error/HTTP error if wrong.