

# COMS E6998: Microservices and Cloud Applications

## *Lecture 2: Microservice and Cloud Application Basics*

Dr. Donald F. Ferguson  
[dff9@columbia.edu](mailto:dff9@columbia.edu)

# Contents

# Contents

- Introduction: Questions? Comments? Discussion?
- (Web) Application Basics
  - Structure.
  - Application design methodologies.
  - Model-View-Controller.
- Let's Design an Application
  - Data Model.
  - User Interface.
  - Microservice Model.
  - REST and Resource Model.
- 1<sup>st</sup> Assignment.

# Introduction

Questions?  
Comments?  
Discussion?

# (Web) Application Basics

# Seeka TV Demo

The screenshot shows the homepage of the Seeka TV website. At the top, there is a navigation bar with a search bar and a user profile icon. Below the header, the main content features a large image of a man and a woman looking at each other. To the left of the image are three award laurels: 'SAN FRANCISCO WEB FEST 2015 NOMINEE', 'PULICENCE CHOICE MIAMI WEB FEST 2016', and 'NOMINEE BEST SUPPORTING ACTRESS AUSTINWEBFEST 2016'. The title 'LOVERS & OTHERS' is prominently displayed in large, bold letters across the center of the image. Below the main image, there is a 'Popular' section featuring thumbnails for several other web series: '20 SECONDS TO LIVE', 'Theater people', 'NUMBER OF SILENCE', 'FEMALE FRIENDS', '# CURRENTLY', 'PROBLEMS', 'RUNNING WITH VIOLET', and 'SILENT'. At the bottom, there is a 'What's New' section with small thumbnail images.

# Seeka TV Demo

The screenshot shows a web browser displaying the Seeka TV website at <https://www.seeka.tv/#/index>. The main content area features a video player with a thumbnail image of a man and a woman. Overlaid on the thumbnail is the title "FIRSTS" in large blue letters, with the subtitle "(because no one ever asks about the second time)" in smaller blue text below it. To the left of the video player is a "Popular" section with several thumbnail images for other shows. To the right is a sidebar with a blue header containing the user's profile picture and the text "ee". The sidebar includes links for "My Profile", "Watch Party", "Watch List", "Support", "Need Help?", and "Sign In".

seeka.TV

Search

ee

My Profile

Watch Party

Watch List

Support

Need Help?

Sign In

Popular

FIRSTS  
(because no one ever asks about the second time)

What's New

7

CourseWorks power... Settings MGRS Coordinates... Center Harbor, NH... body-parser Best Online Course... An interactive, mod... localhost:3000/spa... Benefits- Dell saratoga

COMS E6998 – Microservices and Cloud Applications

Lecture 2: Microservice and Cloud Application Basics

© Donald F. Ferguson, 2017. All rights reserved.

# Seeka TV Demo

The screenshot shows a web browser window for the Seeka TV website. The title bar reads "Watch Seeka TV Web Series - x". The address bar shows a secure connection to https://www.seeka.tv/#/mydashboard/dashabout. The page header features the Seeka TV logo, a search bar, and notification icons. A large video player in the center displays a scene of a frozen lake with snow-covered trees in the background. Below the video, a user profile for Donald Ferguson is shown, including a small photo, his name, and a bio: "I hate HTML and the element <div>. Uploadcare.com is pretty cool! Seeka is Cool". Below the profile are navigation links for "About", "Comments", "Viewed", and "Settings". At the bottom, there are sections for "Watch Parties" and "Watch List", both of which mention the TV show "South of Heaven".

Donald Ferguson

I hate HTML and the element <div>. Uploadcare.com is pretty cool! Seeka is Cool

About Comments Viewed Settings

Watch Parties Watch List

Watchlist makes it easy to keep track of what you want to watch

# Web Application Basic Concepts

1. User performs an action that requires data from a database to be displayed.



2. A request is formed and sent from the client to the web server.



3. The request is processed and the database is queried.



6. Information is displayed to the user.

Application User

5. An appropriate response is generated and sent back.

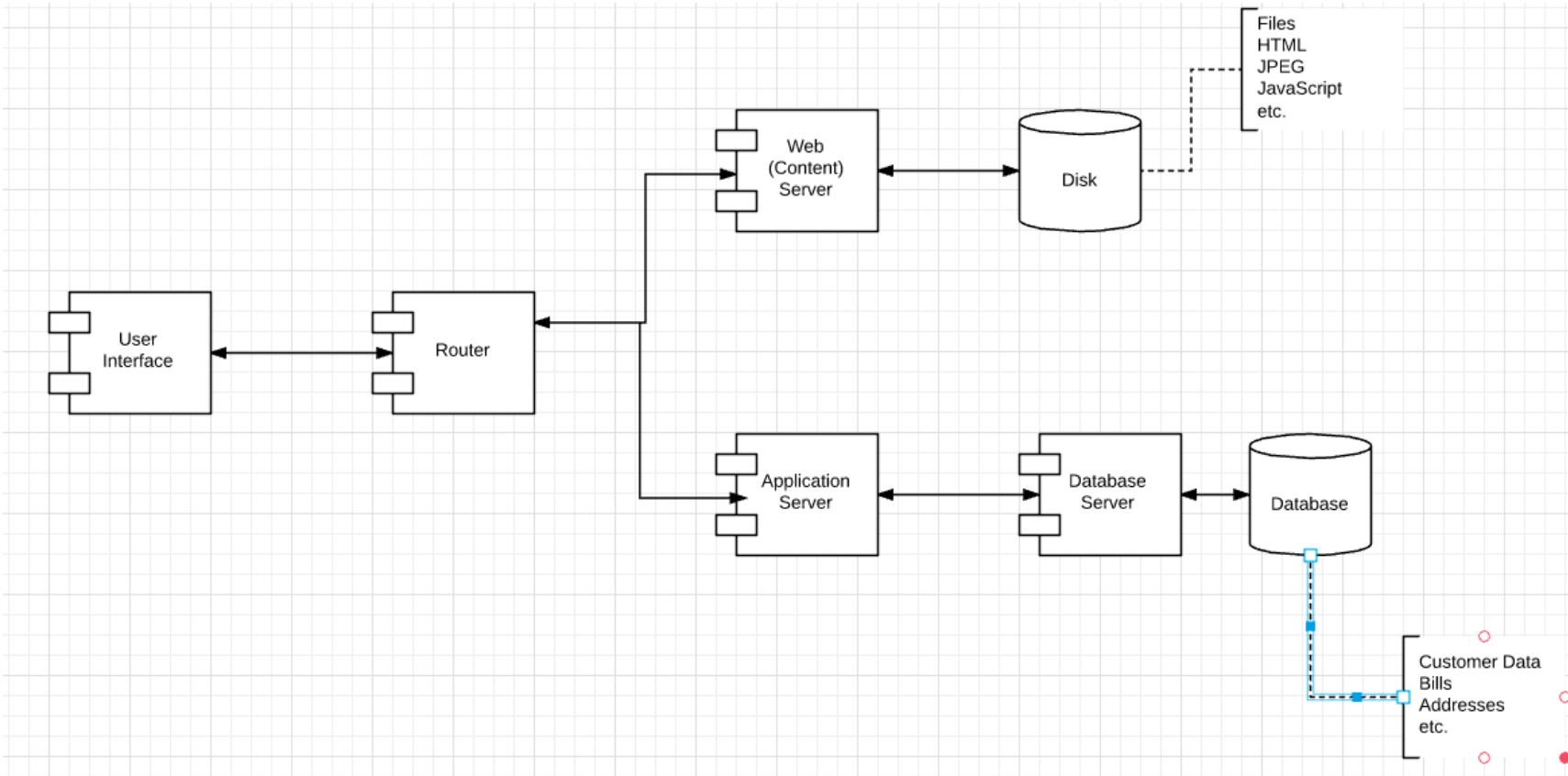
Web Client  
(Presentation Tier)

4. Data is retrieved.

Web Server  
(Application Tier)

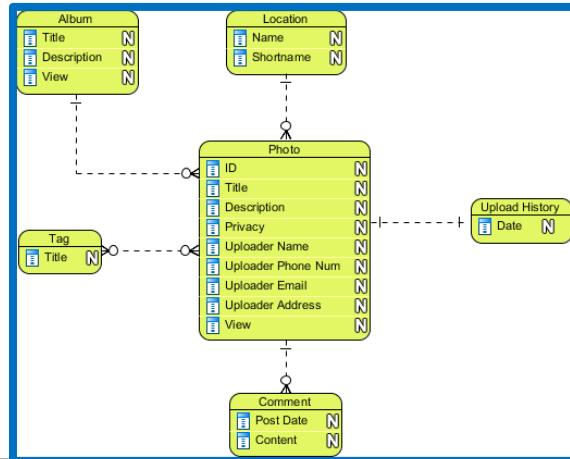
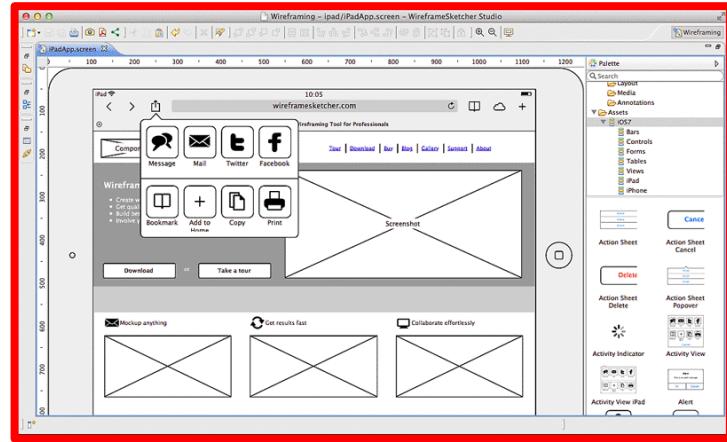
Database  
(Data Tier)

# A “Little More” Detail

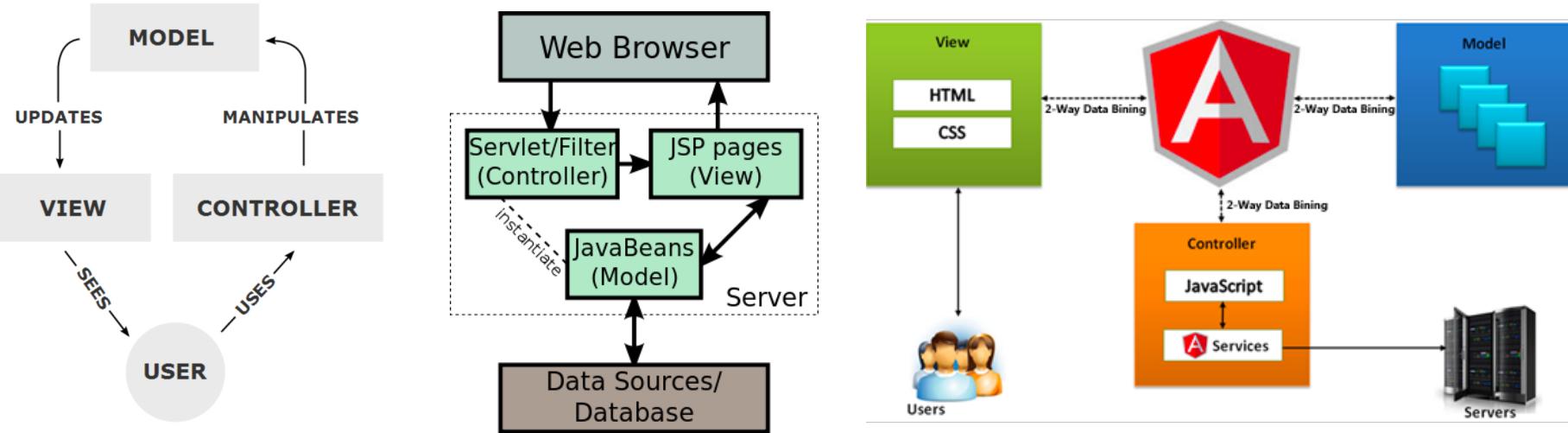


# (Some) Application Design Methodologies

- Start with User Interface
  - Roles and personas.
  - Wireframes for pages and interactions.
  - Page flow/transition diagrams.
- Start with Data Model
  - Entity types, properties.
  - Relationships/associations.
  - Constraints.
  - Operations.



# Model – View – Controller



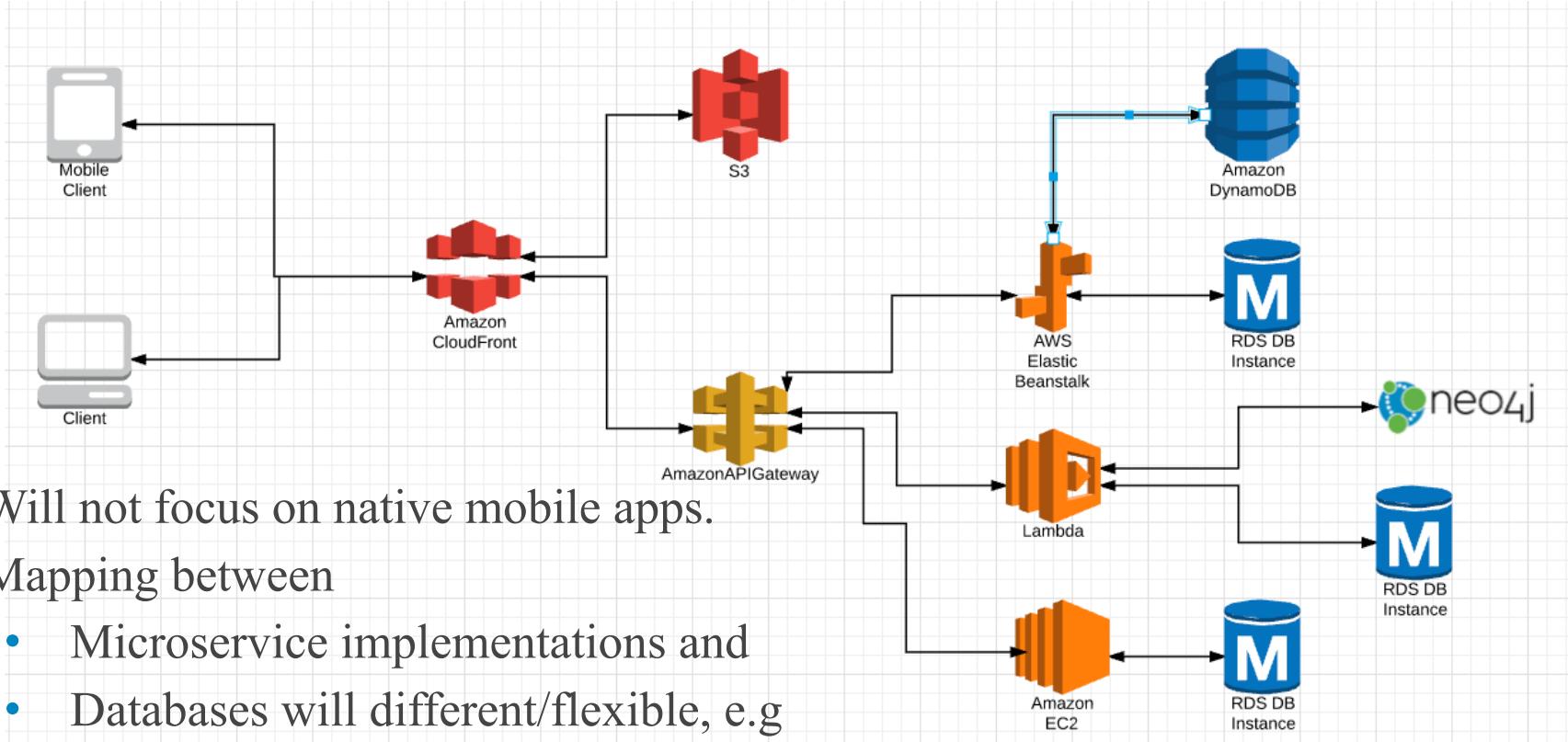
Model – View – Controller is a common paradigm

- Model = Data and application logic
- View = What the user sees and interacts with
- Controller = Maps between user interaction and model, or model changes and UI

# Note

- I know this is elementary.
- Just want to make sure everyone has some basics in common, including terminology.
- Our initial focus will be
  - Data
  - APIs
  - Microservices
- But,
  - Projects and demos will require some basic UIs.
  - UI concepts will become more complex, especially as we add multitenancy.

# Structure of First Couple of Projects

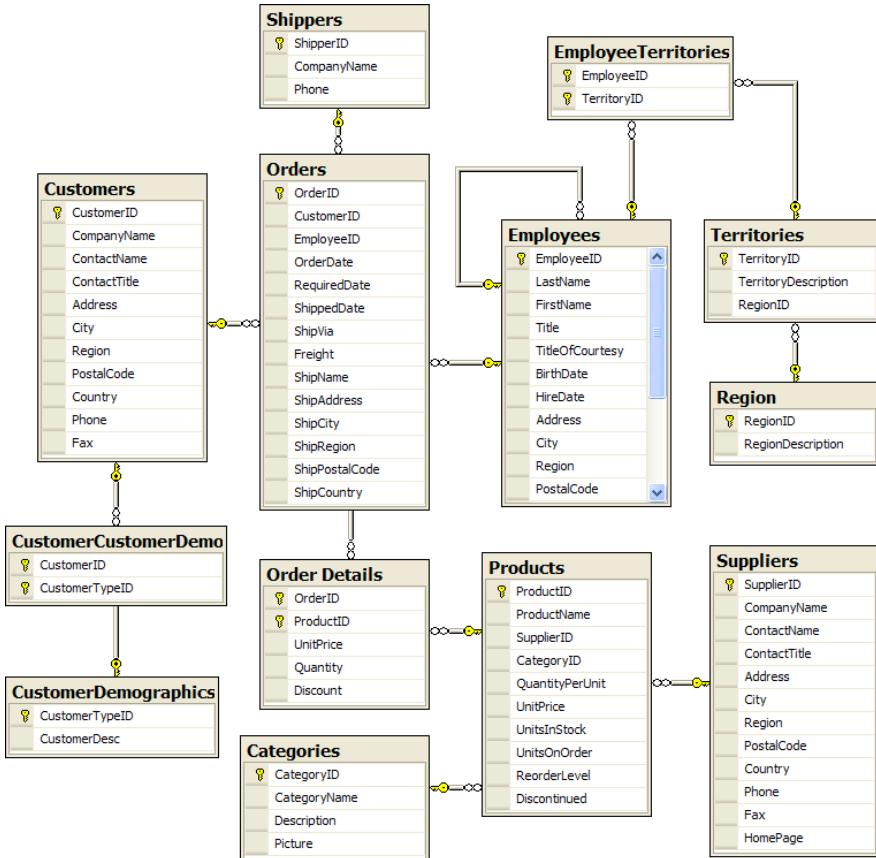


- Will not focus on native mobile apps.
- Mapping between
  - Microservice implementations and
  - Databases will different/flexible, e.g Lambda – Dynamo.

# Let's Design an Application

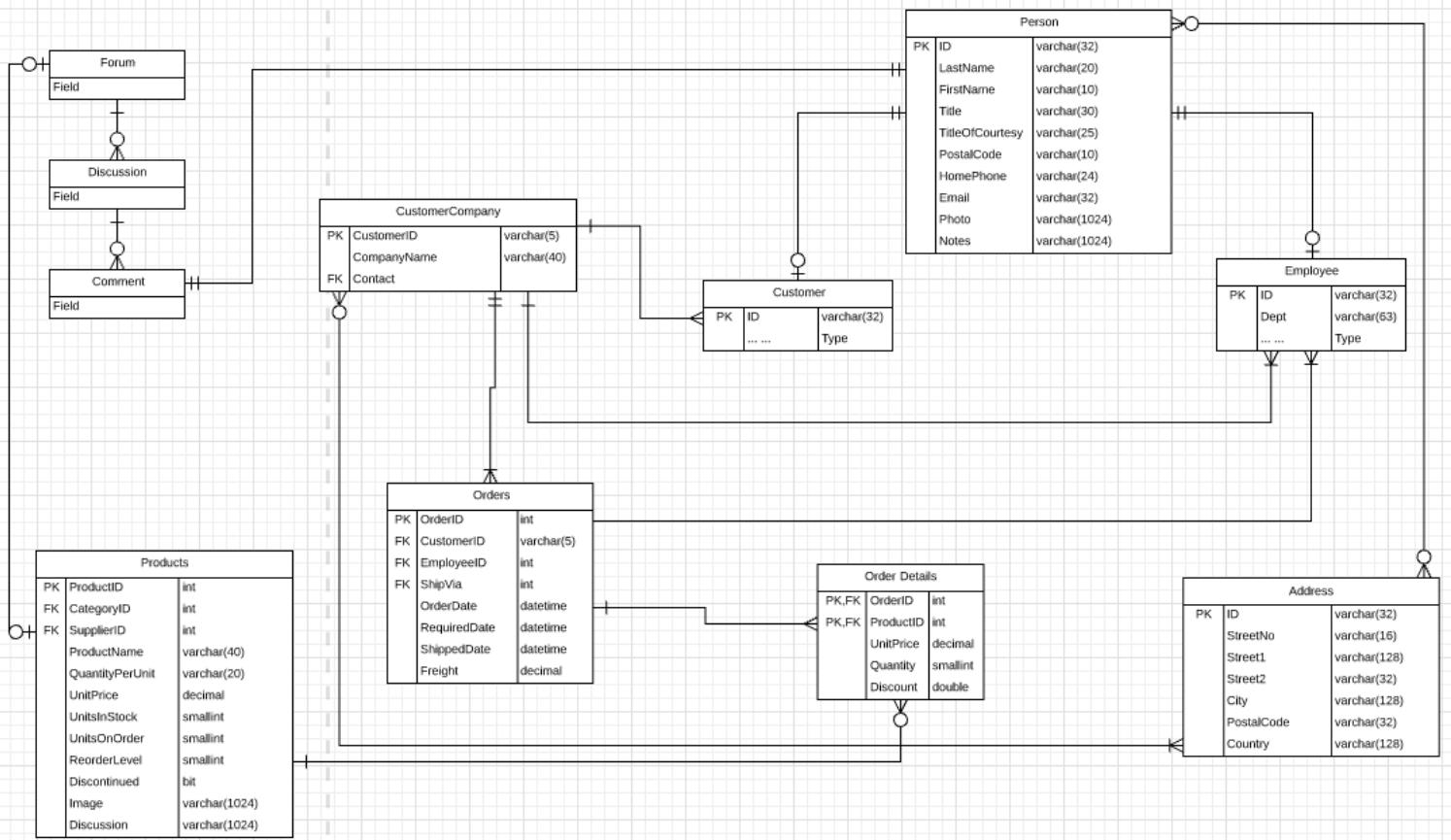
# Datamodel

# Northwind Database (<https://northwinddatabase.codeplex.com/>)



- Relatively simple datamodel/database that
  - Provides core concepts
    - Entity/Resource.
    - Relationships.
  - Sample data for tests and demos.
- Tools and tutorials for
  - Importing in RDS.
  - Importing in DynamoDB.
  - etc.

# Our Modified Data Model



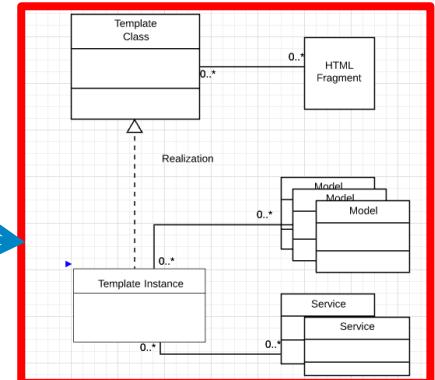
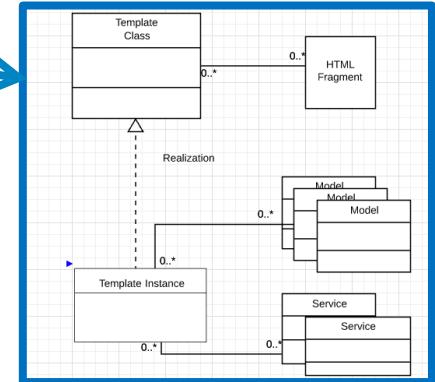
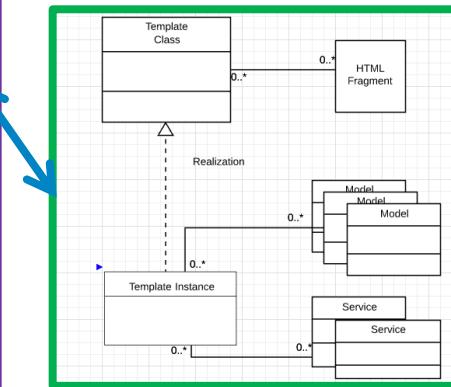
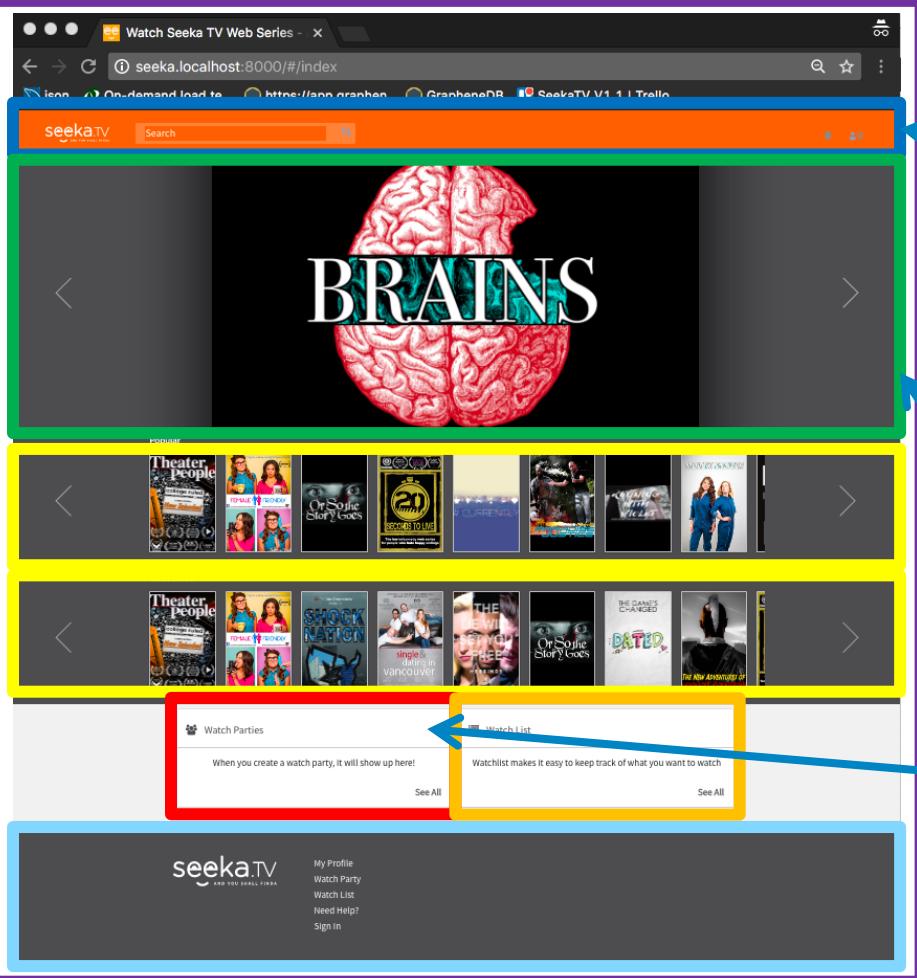
# Motivation

This is not a database or database modeling course. So, why are we implementing this model? It provides a foundation for some interesting use cases:

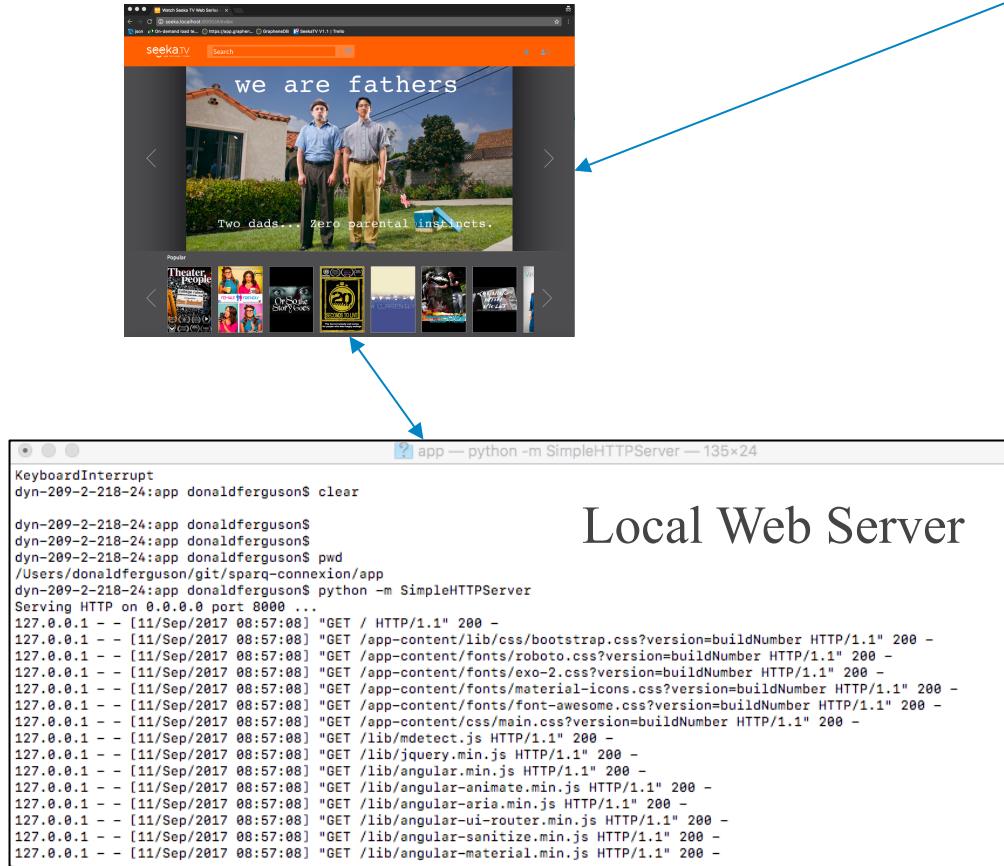
- Model of a resource with properties in a DB and links to web resources, e.g.
  - Photos of people and products.
  - Product discussion forums.
- Designing a good REST API, e.g.
  - /orders/213/orderdetails/456/product
  - /orders?q='CustomerID=123&shipped=false'
  - /orders/123/OrderDetails?f='ProductID,Quantity'
  - Asynchronous methods
- Integration with external cloud services, e.g. address verification, Facebook.
- Handling and representing various types of data
  - Structured.
  - Unstructured.
  - Semi-structured.
- Microservices that: extend and integrate other microservices, implement technical functions

# User Interface

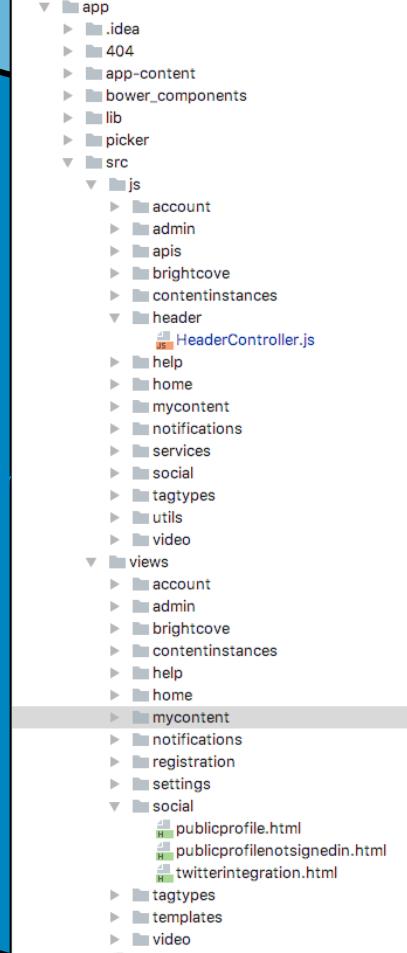
# UI Conceptual Implementation



# Demo Web Content Structure



Local  
File  
System



# Angular JS Material (<https://material.angularjs.org/>)

The screenshot shows the AngularJS Material documentation website at <https://material.angularjs.org/latest/layout/container>. The left sidebar has a blue header "AngularJS Material" with a red hexagon icon containing a white letter "A". Below it are sections: Customization, CSS, Theming, API Reference, Layout (with Introduction), Layout Containers (selected), Layout Children, Child Alignment, Extra Options, Troubleshooting, Services, Types, Directives, and Contributors. The main content area has a header "Layout > Layout Containers". It contains a section titled "Layout and Containers" with a note about the `layout` directive. Below this is a diagram titled "Layout Directive" showing a row with three items: "First item in row" (green), "Second item in row" (purple), "First item in column" (green), and "Second item in column" (purple). A note below the diagram states that `layout` only affects immediate children. The footer includes links to "COMS E6998 – Microservices and Cloud Applications" and "Lecture 2: Microservice and Cloud Application Basics".

- Class examples and lectures will use Angular JS Material.
- UIs will be relatively simple; this is not a UI design and implementation course.
- There are equivalent and/or more modern UI tools.
- Angular JS Material will become useful when we explore multitenancy.

# Angular JS Material (<https://material.angularjs.org/>)

The screenshot shows a browser window with the URL <https://material.angularjs.org/latest/layout/container>. The page title is "Layout > Layout Containers". On the left, there's a sidebar with a large red hexagon icon containing a white letter "A", labeled "AngularJS Material". The sidebar has a tree view with sections like "Customization", "THEMING", "API Reference", "LAYOUT" (which is expanded to show "Introduction", "Layout Containers", "Layout Children", "Child Alignment", "Extra Options", and "Troubleshooting"), "SERVICES", "TYPES", "DIRECTIVES", and "CONTRIBUTORS". The main content area has a blue rounded rectangle background. It contains the heading "UI Implementation Guidance (for class)" and a bulleted list:

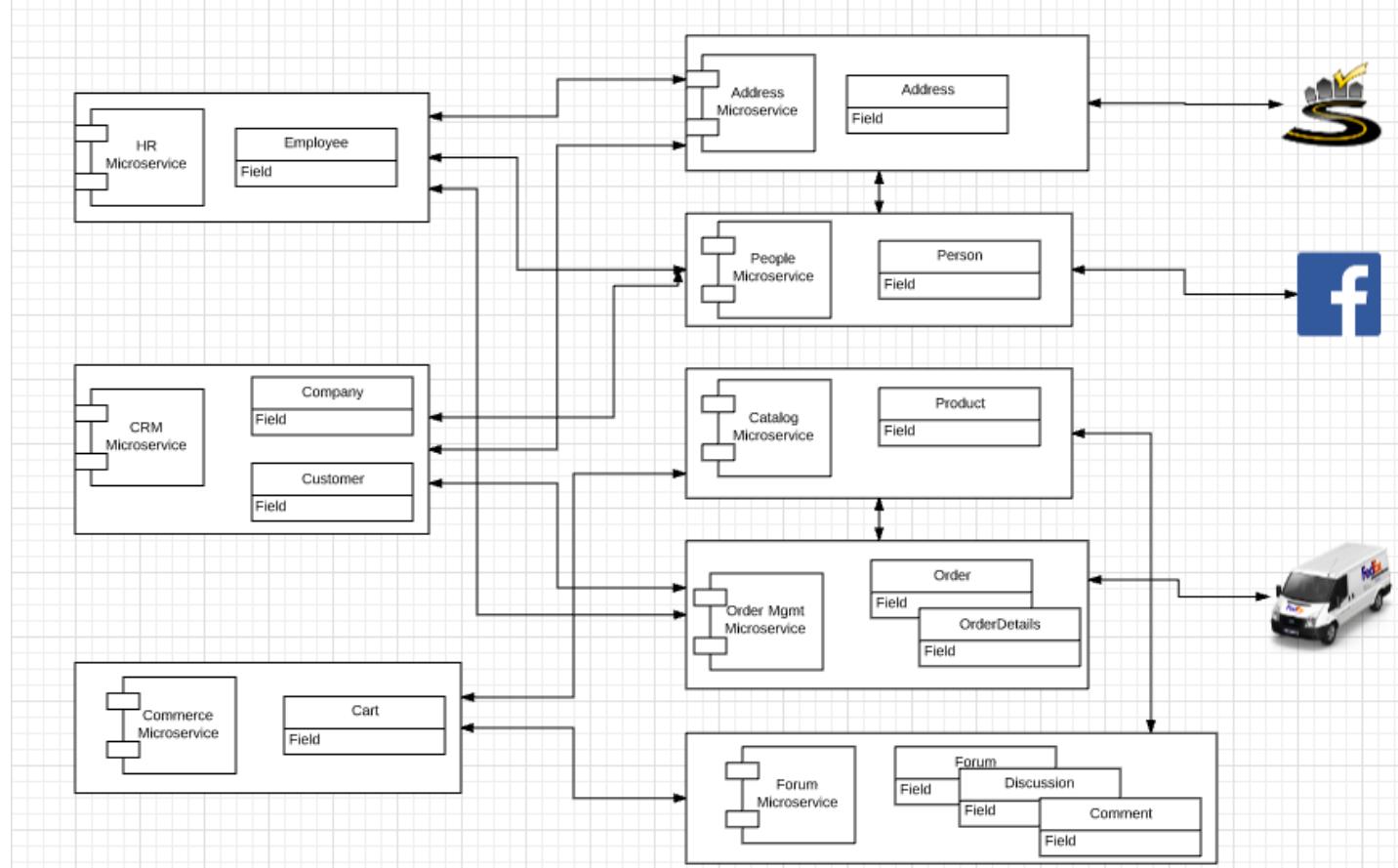
- Simple is good.
- Objectives
  - Understand basic concepts.
  - Provide something for demos and tests.
  - Useful for our introduction/overview of multitenancy.

At the bottom of the main content area, there's a note: "To make your layout automatically change depending upon the device screen size, use one of the following [layout APIs](#) to set the layout direction for devices with view widths:"

- Class examples and lectures will use Angular JS Material.

# Microservice Model

# Microservice Model



# Microservice Model

- Different microservice “instances” will
  - Use different implementation technology, e.g. Lambda, Elastic Beanstalk
  - Use different databases, e.g. RDS, DynamoDB, S3, Neo4J
- The categories of microservices
  - Base services that manage group of related resources, e.g. Product.
  - Services that manage a group of related resources, and integrate with external cloud applications/APIs, e.g.
    - Order Management → FedEx
    - People → Facebook
  - Services that aggregate, compose and orchestrate multiple services into a new, microservice, e.g
    - HR
    - CRM
  - Technical enabling services, e.g. text search, image management.

# REST Model

# Representational State Transfer (REST)

- People confuse
  - Various forms of RPC/messaging over HTTP
  - With REST
- REST has six core tenets
  - Client/server
  - Stateless
  - Caching
  - **Uniform Interface**
  - Layered System
  - Code on Demand

*HATEOAS: Hypertext as the Engine of Application State --*  
The principle is that a client interacts with a network application entirely through [hypermedia](#) provided dynamically by application servers. A REST client needs no prior knowledge about how to interact with any particular application or server beyond a generic understanding of hypermedia.

# REST Tenets

- Client/Server (Obvious)
- Stateless is a bit confusing
  - The server/service maintains *resource* state, e.g. Customer and Agent info.
  - The *conversation* is stateless. The client provides all conversation state needed for an API invocation. For example,
    - `customerCursor.next(10)` requires the *server* to remember the client's position in the iteration through the set.
    - A *stateless* call is `customerCollection.next("Bob", 10)`. Basically, the client passes the cursor position to the server.
- Caching
  - The web has significant caching (in browser, CDNs, ...)
  - The resource provider must
    - Consider caching policies in application design.
    - Explicitly set control fields to tell clients and intermediaries what to cache/when.

# REST Tenets

- Uniform Interface
  - Identify/locate resources using URIs/URLs.
  - A fixed set of “methods” on resources.
    - myResource.deposit(21.13) is not allowed.
    - The calls are
      - Get
      - Post
      - Put
      - Delete
  - Self-defining MIME types (Text, JSON, XML, ...).
  - Default web application for using the API.
  - URL/URI for relationship/association.
- Layered System: Client cannot tell if connected to the server or an intermediary performing value added functions, e.g.
  - Load balancing.
  - Security.
  - Idempotency.
- Code on Demand (optional): Resource Get can deliver helper code, e.g.
  - JavaScript
  - Applets

# Anatomy of a URL and Methods (not *normative*)

- Collection/Set: /customers
  - POST: Create a new customer
  - GET: Return all customers, and implement
    - Query
    - Pagination
- Specific Instance: /customers/dff9 (dff9 is a unique, primary key)
  - GET: Some or all fields.
  - PUT: Update some or all fields.
  - DELETE: Delete the specific customer.
- Relationships:
  - Set: /customers/dff9/orders
  - Instance: /customers/dff9/orders/21

# Success Response Codes

Operation	HTTP Request	HTTP Response Codes Supported
READ	GET	200 - OK with message body 204 - OK no message body 206 - OK with partial message body
CREATE	POST	201 - Resource created (Operation Complete) <b>202 - Resource accepted (Operation Pending)</b>
UPDATE	PUT	<b>202 - Accepted (Operation Pending)</b> 204 - Success (Operation Complete)
DELETE	DELETE	<b>202 - Accepted (Operation Pending)</b> 204 - Success (Operation Complete)

- 202 means
- Your request went asynch.
  - The HTTP header Link is where to poll for rsp.
  - We will cover later.

Examples of Link Headers in HTTP response:

```
Link: <http://api/jobs/j1>;rel=monitor;title="update profile"  
Link: <http://api/reports/r1>;rel=summary;title="access report"
```

# (Some) Failure Response Code

Error	Response Code
Invalid Parameter	400 - Invalid parameter
Authentication	401 - Authentication failure
Permission Denied	403 - Permission denied
Not Found	404 - Resource not found
Invalid Request Method	405 - Invalid request method
Unprocessable Entity	422 – Unprocessable Entity
Internal Server Error	500 - Internal Server Error
Service Unavailable	503 - Service Unavailable

# HATEOAS Response Format

Person		
PK	ID	varchar(32)
	LastName	varchar(20)
	FirstName	varchar(10)
	Title	varchar(30)
	TitleOfCourtesy	varchar(25)
	HomePhone	varchar(24)
	Email	varchar(32)
	Photo	varchar(1024)
	Notes	varchar(1024)
	Address	varchar(1024)

- Relationships
  - Return/Send a URL *link* to the related resource.
  - Not the piece of data that is part of the link.
- Returning some of the data and a link is OK.
- There is no single, agreed best practice, but most good implementations do some version of this pattern.
- There are some standard link types, but not all apply.  
(<https://www.iana.org/assignments/link-relations/link-relations.xhtml>)

```
{  
    id: "dff9",  
    lastName: "Ferguson",  
    firstName: "Donald",  
    title: "Professor",  
    titleOfCourtesy: "Dr.",  
    email: "dff9@columbia.edu",  
    phone: "212-555-1212",  
    links: {  
        address: {  
            href: "/address/1234"  
        },  
        self: {  
            href: "/Persons/dff9"  
        },  
        photo: {  
            href: "https://static1.squarespace.com/static/57fd476a197aea31fab38eed/t/57fe5a3a20099e37a0754ccd/1476287038972/don.png?format=300"  
        }  
    }  
}
```

# Handling Links

- In a Customer
  - The address ID
  - Is actually a *link*
  - Into another set of *resources*

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

- Resolving the link would require

- Non-hypertext/web
- Side knowledge
- And violate HATEOAS
- And forces some out-of-base documentation/info.

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

- Instead, return links as links

# Several Patterns, for Example

## Simple

HTTP/1.1 200 OK  
Content-Type: application/json; charset=UTF-8

```
{
  "href" : "https://api.stormpath.com/v1/accounts/cJoiwcorTTmkDDBsf02AbA",
  "username" : "jlpicard",
  "email" : "capt@enterprise.com",
  "givenName" : "Jean-Luc",
  "middleName" : "",
  "surname" : "Picard",
  "status" : "enabled",
  "directory" : {
    "href" : "https://api.stormpath.com/v1/directories/WpM9nyZ2TbaEzfbRvLk9K"
  },
  ...
}
```

Link with Resource Expansion

```
{
  "href": "https://api.stormpath.com/v1/accounts/ZugcG3JHQFOTKGEXAMPLE",
  "username": "lonestarr",
  "email": "lonestarr@druidia.com",
  "fullName": "Lonestarr Schwartz",
  "givenName": "Lonestarr",
  "middleName": "",
  "surname": "Schwartz",
  "status": "ENABLED",
  "emailVerificationToken": null
  "directory" : {
    "href": "https://api.stormpath.com/v1/directories/S2Hzc7gXTumVYEXAMPLE",
    "name": "Spaceballs",
    "description": "",
    "status": "ENABLED",
    "accounts": {
      "href": "https://api.stormpath.com/v1/directories/S2Hzc7gXTumVYEXAMPLE/accounts"
    },
    "groups": {
      "href": "https://api.stormpath.com/v1/directories/S2Hzc7gXTumVYEXAMPLE/groups"
    },
    "tenant":{
      "href": "https://api.stormpath.com/v1/tenants/wGbGaSNuTUix9EXAMPLE"
    }
  },
  "tenant" : {
    "href": "https://api.stormpath.com/v1/tenants/wGbGaSNuTUix9EXAMPLE"
  },
  ...
}
```

Data fields followed by links section.

```
{
  "links": [
    { "rel" : "directory", "link" : "https://api.stormpath.com/v1/directories/WpM9nyZ2TbaEzfbRvLk9" },
    { "rel" : "tenant", "link" : "https://api.stormpath.com/v1/tenants/wGbGaSNuTUix9" }
  ],
  ...
}
```

# Pagination/Iteration

- Consider the web method GET on
  - /customers?lastname=Smith
  - /tvshows?year=2016
- The result set could be thousands or millions of “records.”
- The application should not/cannot return all records at once
  - Network/connection timeouts.
  - Overwhelm the client runtime with data and data processing.
  - Give the user a chance to realize
    - That they asked the “wrong question”
    - And submit a refined query.
  - etc.

# Traditional Solution is Iterator/Cursor

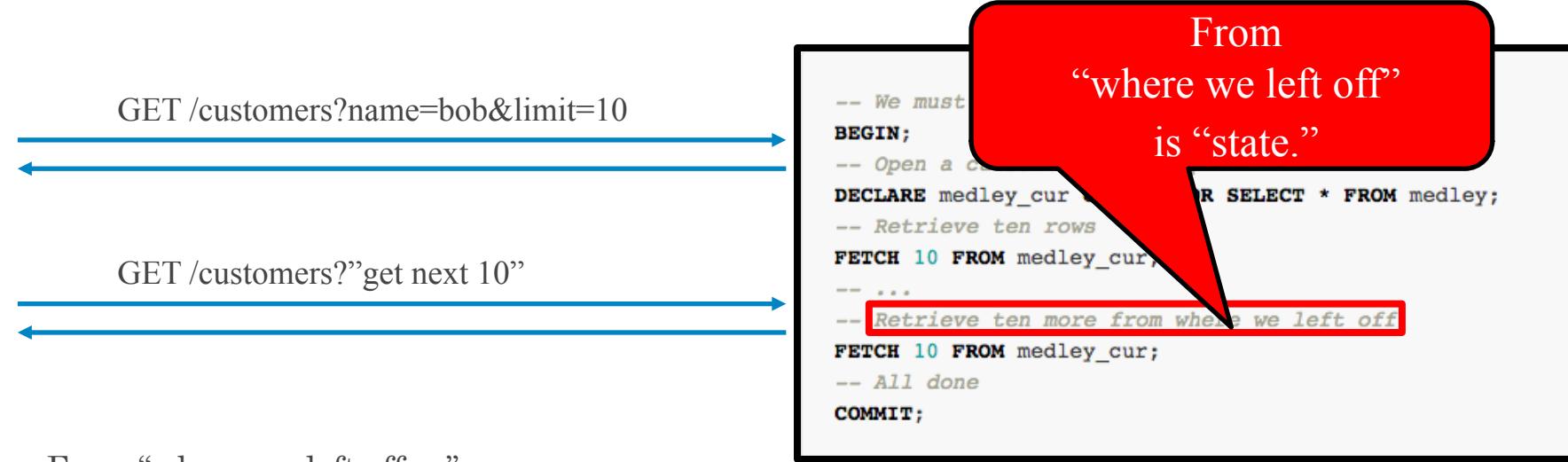
GET /customers?name=bob&limit=10



```
-- We must be in a transaction
BEGIN;
-- Open a cursor for a query
DECLARE medley_cur CURSOR FOR SELECT * FROM medley;
-- Retrieve ten rows
FETCH 10 FROM medley_cur;
-- ...
-- Retrieve ten more from where we left off
FETCH 10 FROM medley_cur;
-- All done
COMMIT;
```

But this requires conversation state!

# Traditional Solution is Iterator/Cursor



- From “where we left off ...”
  - Requires client specific state held between requests, which violates
  - The REST tenet of “stateless.”
- The benefits of stateless are
  - Not holding server resources (cursors, memory, ...) for millions of clients for several seconds/client.
  - Exploits the computing and storage of millions of client “computers.”
  - And Lambda functions are STATELESS by definition. **So, you have to do it.**

# Examples

- SQL
  - URLs
    - GET /customers?name=Bob&limit=10.
    - GET /customers?name=Bob&limit=10&offset=10
  - SQL
    - SELECT \* FROM customers WHERE name='Bob' LIMIT 10;
    - SELECT \* FROM customers WHERE name='Bob' LIMIT 10 OFFSET 10;
- DynamoDB –
  - SELECT \* → AttributesToGet : [ ... ]
  - WHERE → KeyConditions: [ ... ]
  - LIMIT → Limit:
  - OFFSET → ExclusiveStartKey ==
    - The **LastEvaluatedKey** of the last record in the **returned** previous query result set.
    - Start the query at the record after the ExclusiveStartKey

# Example

Next Request

Previous Response

```
  "S": "string"
}
},
"LastEvaluatedKey": {
  "string" : {
    "B": blob,
    "BOOL": boolean,
    "BS": [ blob ],
    "L": [
      "AttributeValue"
    ],
    "M": {
      "string" : "AttributeValue"
    },
    "N": "string",
    "NS": [ "string" ],
    "NULL": boolean,
    "S": "string",
    "SS": [ "string" ]
  }
},
"ScannedCount": number
}
```

```
var params = {
  TableName: 'table_name',
  IndexName: 'index_name', // optional (if querying an index)
  KeyConditions: { // indexed attributes to query
    // must include the hash key value of the table or index
    // with 'EQ' operator
    attribute_name: {
      ComparisonOperator: 'EQ', // (EQ | NE | IN | LE | LT | GE | GT | BETWEEN |
                                // NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH)
      AttributeValueList: [ { S: 'STRING_VALUE' }, ],
    },
    // more key conditions ...
  },
  ScanIndexForward: true, // optional (true | false) defines direction of Query in the index
  Limit: 0, // optional (limit the number of items to evaluate)
  ConsistentRead: false, // optional (true | false)
  Select: 'ALL_ATTRIBUTES', // optional (ALL_ATTRIBUTES | ALL_PROJECTED_ATTRIBUTES |
                            // SPECIFIC_ATTRIBUTES | COUNT)
  AttributesToGet: [ // optional (list of specific attribute names to return)
    'attribute_name',
    // ... more attributes ...
  ],
  ExclusiveStartKey: { // optional (for pagination, returned by prior calls as LastEvaluatedKey)
    attribute_name: { S: 'STRING_VALUE' },
    // anotherKey: ...
  },
  ReturnConsumedCapacity: 'NONE', // optional (NONE | TOTAL | INDEXES)
};

dynamodb.query(params, function(err, data) {
  if (err) console.log(err); // an error occurred
  else console.log(data); // successful response
});
```

# Pagination –

The response to the client has info about “where you left off” and client returns it.

```
{"data":  
[{"user_id": "42", "name": "Bob",  
"links": [{"rel": "self", "href": "http://api.example.com/users/42"}]},  
 {"user_id": "22", "name": "Frank",  
"links": [{"rel": "self", "href": "http://api.example.com/users/22"}]},  
 {"user_id": "125", "name": "Sally",  
"links": [{"rel": "self", "href": "http://api.example.com/users/125"}]},  
 "links":  
[{"rel": "first", "href": "http://api.example.com/users?offset=0&limit=3"},  
 {"rel": "last", "href": "http://api.example.com/users?offset=55&limit=3"},  
 {"rel": "previous", "href": "http://api.example.com/users?offset=3&limit=3"},  
 {"rel": "next", "href": "http://api.example.com/users?offset=9&limit=3"}]}
```

# Pagination –

The response to the client has to return:

```
{"data":  
[{"user_id": "42", "name": "Bob",  
"links": [{"rel": "self", "href": "http://api.example.com/users/42"}],  
 {"user_id": "22", "name": "Frank",  
"links": [{"rel": "self", "href": "http://api.example.com/users/22"}]},  
 {"user_id": "125", "name": "Sally",  
"links": [{"rel": "self", "href": "http://api.example.com/users/125"}]}],  
"links":
```

```
[{"rel": "first", "href": "http://api.example.com/users?offset=0&limit=3"},  
 {"rel": "last", "href": "http://api.example.com/users?offset=55&limit=3"},  
 {"rel": "previous", "href": "http://api.example.com/users?offset=3&limit=3"},  
 {"rel": "next", "href": "http://api.example.com/users?offset=9&limit=3"}]}
```

The response contains

- First, Last, Previous, Next
- With hrefs and URLs that are “opaque” strings.

So, the client is isolated

- From the specific DB in use.
- From forming database specific strings.

# REST and Query

REST URLs surface two types of “query:”

## 1. Simple

- /customers?lastName=Ferguson&title=Professor
- Logically maps to SQL
  - SELECT \* FROM Customers WHERE lastName='Ferguson' AND title='Professor'
- But you may have to
  - Map DB column names to more intuitive REST property names.
  - Map from SQL (or other types) to valid JSON or HTML types.

## 2. Complex: Standard HTTP does not support OR, <, >=, LIKE, ...

- Use the q=“ approach: /customers?q=“lastName LIKE (Ferg) AND IQ<40”
- Must map your logical language into the microservice/DB specific language.
- May have to URL/Base64 encode the query string.

In either case, you have to parse and validate the string, e.g. to prevent *injection attacks* of the form:

/customers?lastName=Ferguson OR (1=1)

# Swagger and Swagger Hub

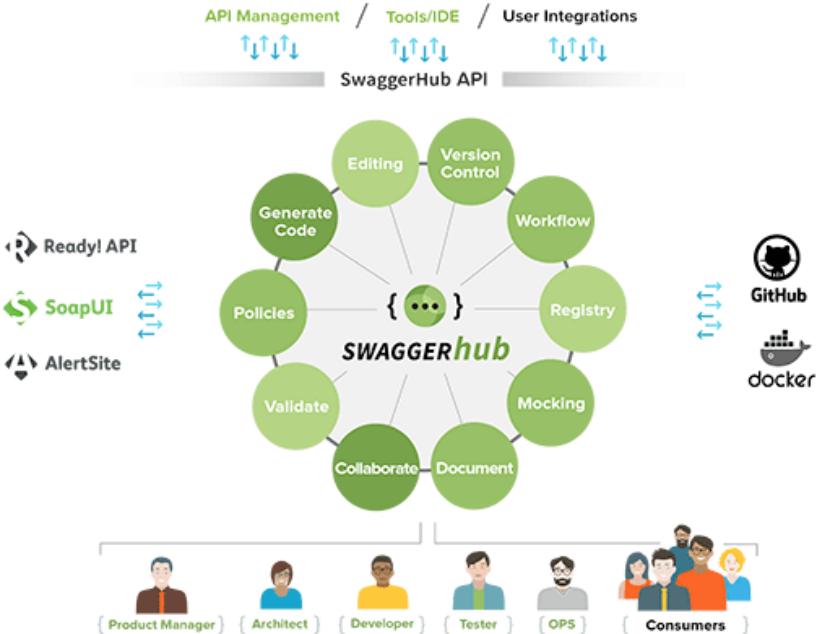
## Design • Document • Discover

With SwaggerHub, you have the whole API Lifecycle at your fingertips:

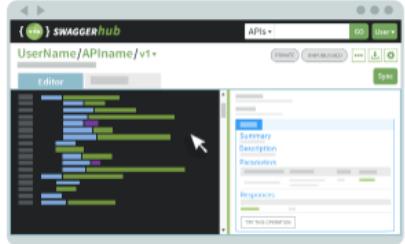
- Collaborative design.
- Interactive documentation.
- And an easily searchable registry of Swagger-based APIs.

At its core, SwaggerHub is based on the Swagger principles of open, integrated technologies.

- Integrate with GitHub to protect your API versions.
- Link to DockerHub so API consumers can easily download and use your API.
- Use the SwaggerHub Registry API to integrate with our API directory.

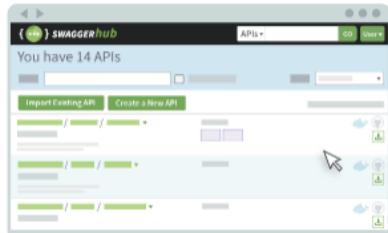


## Definition Editor



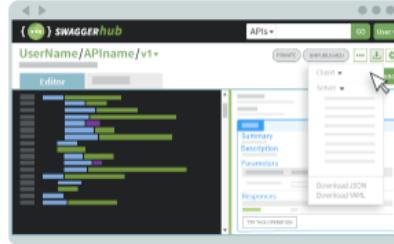
Use our intuitive editors to create your API definition and collaborate with others on their API definitions. On-the-fly validation keeps you honest.

## API Registry



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

## Code Gen



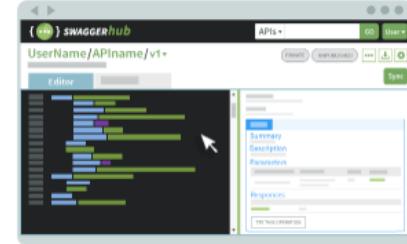
Get your development project off to a quick start by using our client and server code templates.

## Domains



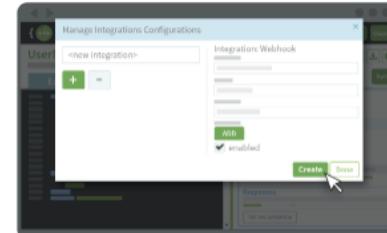
Store all your re-usable components that can later be used across multiple API definitions, saving you time

## Versioning



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

## Plugins



## LEARN MORE

# Swagger Editor

The screenshot shows the Swagger Editor interface with the following components:

- Header:** SWAGGER hub, / apis / donald.fer... / Test / 1.0.0-, PUBLIC, UNPUBLISHED, VALID, Save.
- Left Sidebar:** Editor, Split, UI, +, Home, Q, SE.
- Code Editor (Editor Tab):** Displays the Swagger JSON definition for the Petstore server. The code is as follows:

```
1 ---  
2 swagger: "2.0"  
3 info:  
4   description: "This is a sample server Petstore server. You can find out  
      more about\\  
5     \\ Swagger at [http://swagger.io](http://swagger.io) or on [irc.freenode  
        .net, #swagger](http://swagger.io/irc/).\\  
6     \\ For this sample, you can use the api key `special-key` to test the  
        authorization\\  
7       \\ filters."  
8   version: "1.0.0"  
9   title: "Swagger Petstore"  
10  termsOfService: "http://swagger.io/terms/"  
11  contact:  
12    email: "apiteam@swagger.io"  
13  license:  
14    name: "Apache 2.0"  
15    url: "http://www.apache.org/licenses/LICENSE-2.0.html"  
16  host: "petstore.swagger.io"  
17  basePath: "/v2"  
18  tags:  
19    - name: "pet"  
20      description: "Everything about your Pets"  
21      externalDocs:  
22        description: "Find out more"  
23        url: "http://swagger.io"  
24    - name: "store"  
25      description: "Access to Petstore orders"  
26    - name: "user"  
27      description: "Operations about user"  
28      externalDocs:  
29        description: "Find out more about our store"  
30        url: "http://swagger.io"  
31  schemes:
```

- Info Panel:** This is a sample server Petstore server. You can find out more about Swagger at <http://swagger.io> or on [irc.freenode.net, #swagger](#). For this sample, you can use the api key `special-key` to test the authorization filters.
- Links:** Terms of service, Contact the developer, Apache 2.0, Find out more about Swagger.
- Schemes:** HTTP, Authorize.
- API Endpoints (pet):**
  - POST /pet**: Add a new pet to the store.
  - PUT /pet**: Update an existing pet.
  - GET /pet/findByStatus**: Finds Pets by status.
  - GET /pet/findByTags**: Finds Pets by tags.

# Swagger Editor

The screenshot shows the Swagger Editor interface. At the top, there's a navigation bar with a green icon, the text "SWAGGER hub", and a dropdown for "dональд.фергюсон". Below the navigation bar, the URL is "/apis / donald.fer... / Test / 1.0.0 -". On the left, there are several icons: a back arrow, a plus sign, a house, a magnifying glass, and a gear. The main area has tabs for "Editor" (selected), "Split", and "UI". The "Editor" tab displays a JSON configuration file for a Petstore server. The "UI" tab shows a sample response from the server, which includes a message about the Petstore server and a link to the Swagger documentation. A large blue callout bubble with the word "Demo" inside points to the "UI" tab. On the right side, there are buttons for "PUBLIC", "UNPUBLISHED", "VALID", "Save", and "Last Saved: 08:56:57 am Sep 29, 2016". Below the UI response, there are three API operations listed: "PUT /pet Update an existing pet", "GET /pet/findByStatus Finds Pets by status", and "GET /pet/findByTags Finds Pets by tags". Each operation has a lock icon to its right.

```
1 ---  
2 swagger: "2.0"  
3 info:  
4   description: "This is a sample server Petstore server. You can find out  
      more about\\  
5      \\ Swagger at  
       .net, #swag  
6      \\ For this:  
        authorization  
7      \\ filters."  
8   version: "1.0"  
9   title: "Swagger Petstore API"  
10  termsOfService: "http://swagger.io/terms.html"  
11  contact:  
12    email: "apiteam@swagger.io"  
13  license:  
14    name: "Apache 2.0"  
15    url: "http://www.apache.org/licenses/LICENSE-2.0.html"  
16  host: "petstore.swagger.io"  
17  basePath: "/v2"  
18  tags:  
19    - name: "pet"  
20      description: "Operations about pet management"  
21      externalDocs:  
22        description: "Find out more"  
23        url: "http://swagger.io"  
24    - name: "store"  
25      description: "Access to Petstore orders"  
26    - name: "user"  
27      description: "Operations about user"  
28      externalDocs:  
29        description: "Find out more about our store"  
30        url: "http://swagger.io"  
31  schemes:
```

This is a sample server Petstore server. You can find out more about Swagger at <http://swagger.io> or on [irc.freenode.net, #swagger](#). For this sample, you can use the api key `special-key` to test the authorization filters.

Demo

PUT /pet Update an existing pet

GET /pet/findByStatus Finds Pets by status

GET /pet/findByTags Finds Pets by tags

# 1<sup>st</sup> Project

## Part 1

# 1st Project – Part 1

- Implement two distinct microservices
  - Person
  - Address
- Tasks
  - Use Swagger Editor to define and document REST APIs.
  - Implement an Elastic Beanstalk application (microservice) for each resource that implements the relevant REST API.
  - Each microservice should support
    - GET and POST on resource, e.g. /Person
    - GET, PUT, DELETE on resource/id, e.g. /Person/dff9
    - Simple query, e.g. /Person?lastName=Ferguson
    - Pagination
    - Relationship paths: /Person/dff9/address and /Addresses/someID/persons
    - HATEOAS links where appropriate.
  - Simple HTML/Angular demo UI.
- Due: 11:59 PM on 26-Sep-2017



- Simple HTML/Angular demo UI
- Due: 11:59 PM on 26-Sep-2017

- This assignment is
  - Complex and a lot of work
  - Using technologies you have just been introduced to.
- I do not expect you to get it all done or get it all correct.
- We will learn iteratively based on feedback and corrections.
- The IAs and I will help.