COMS E6998: **Microservices and Cloud Applications**

Lecture 2: Microservice and Cloud Application Basics

Dr. Donald F. Ferguson
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.
- 1st Assignment.

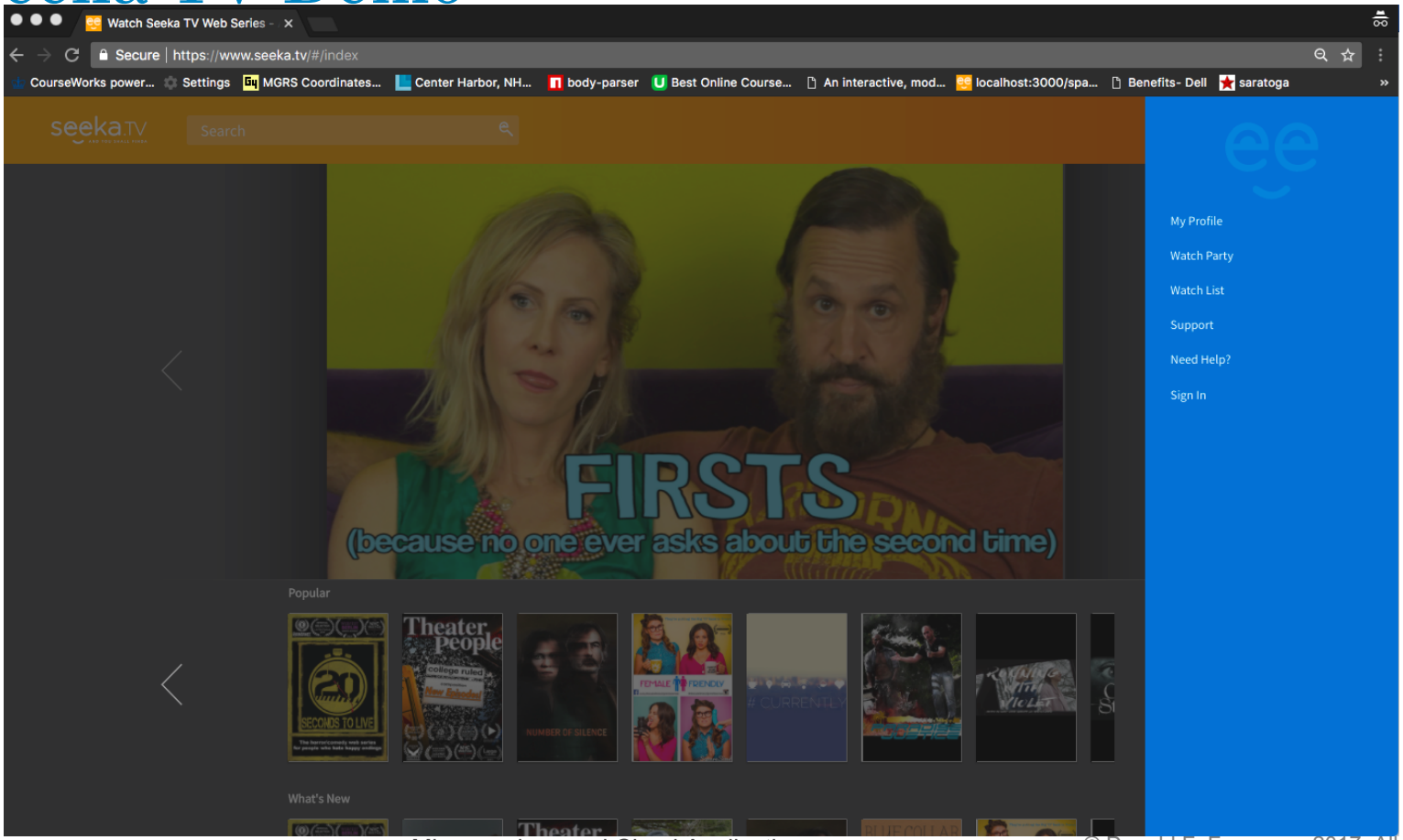
Introduction

Questions?
Comments?
Discussion?

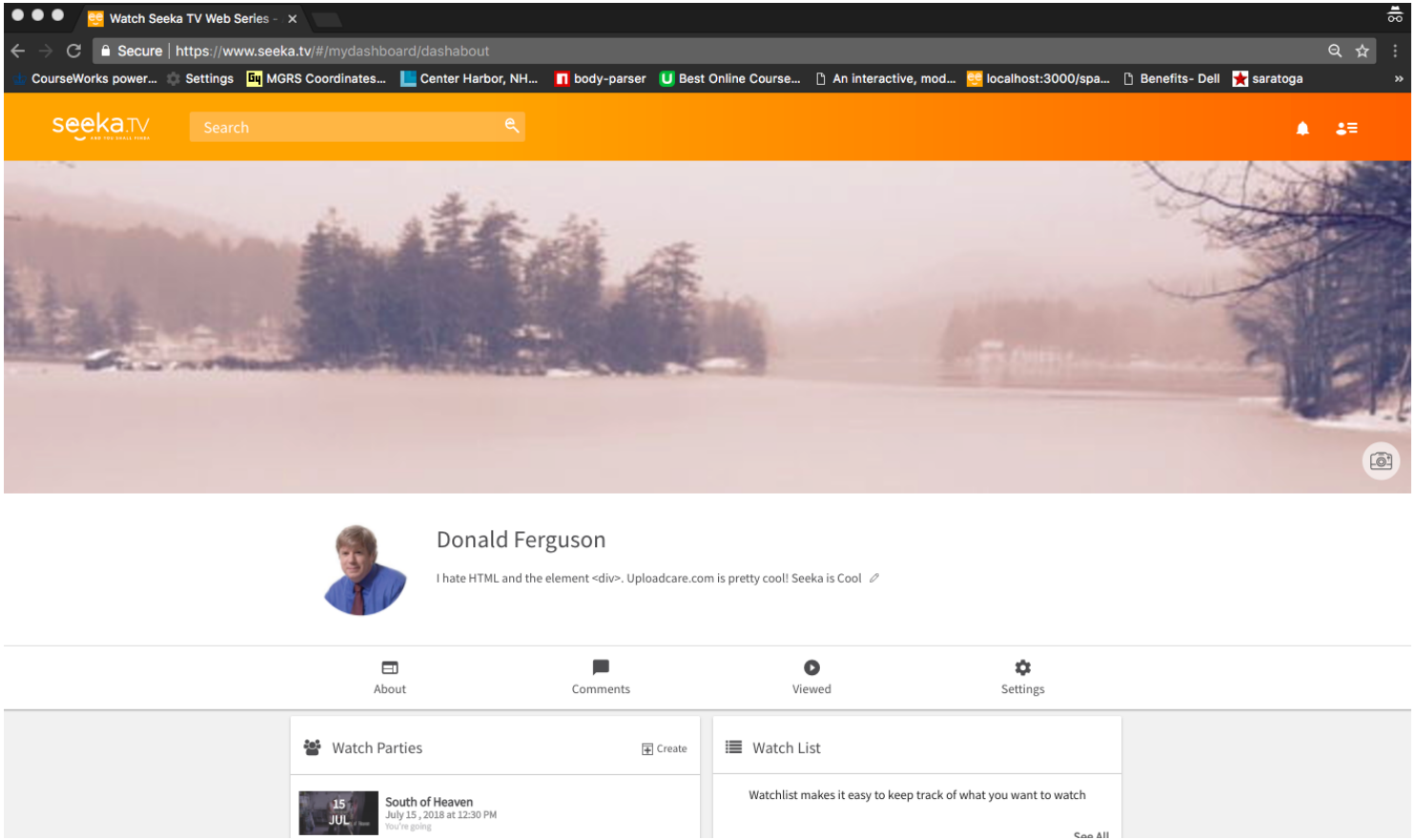
(Web) Application Basics

Seeka TV Demo

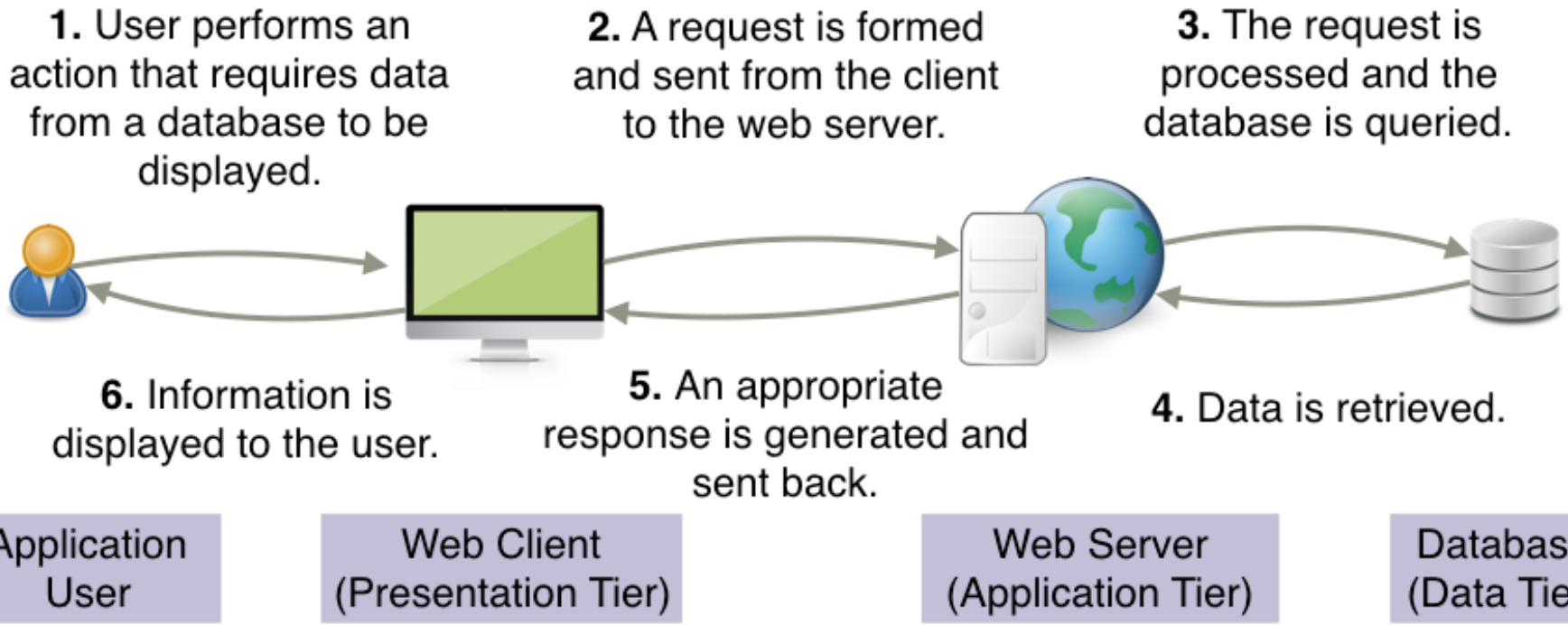
Seeka TV Demo



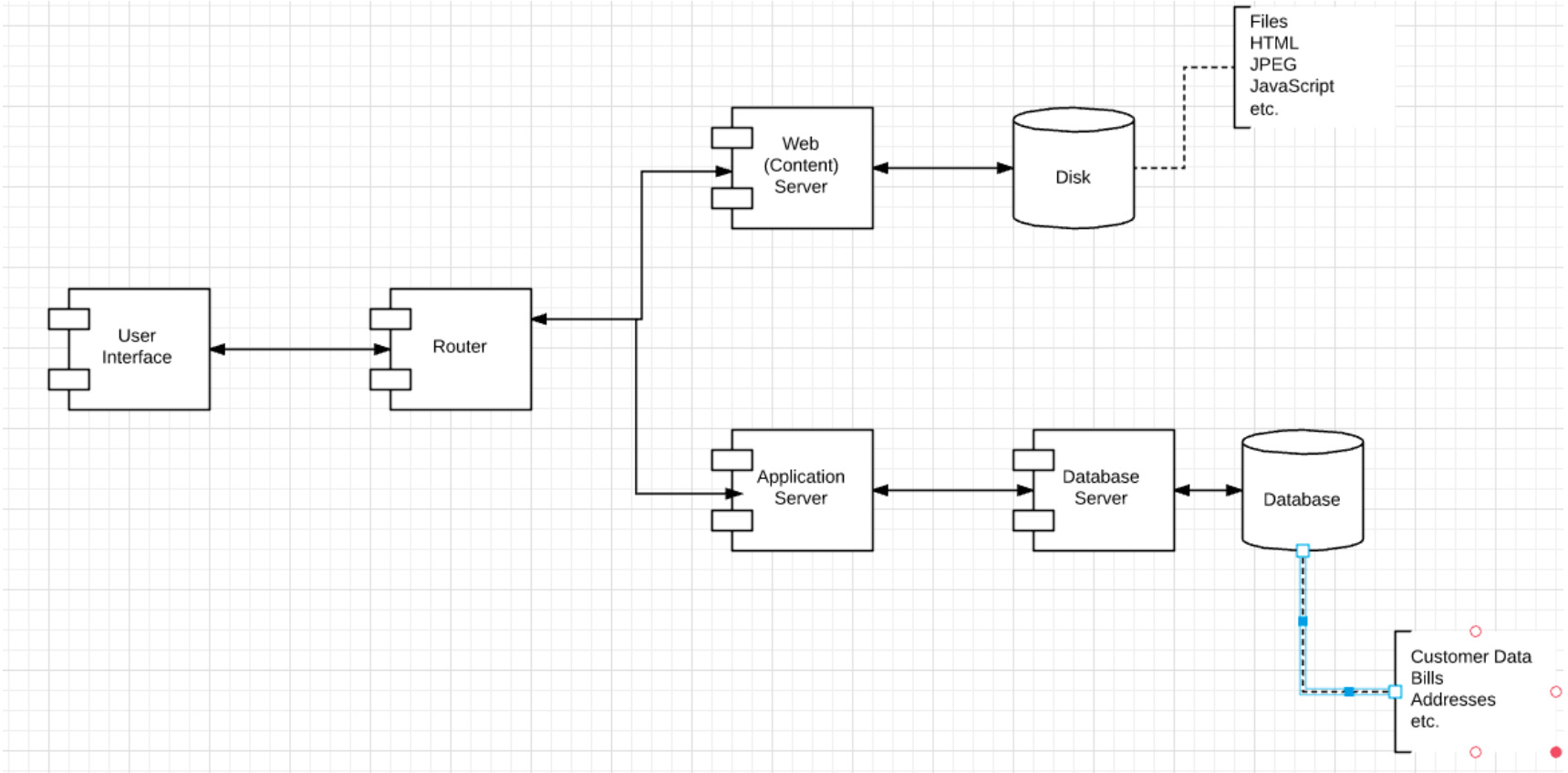
Seeka TV Demo



Web Application Basic Concepts

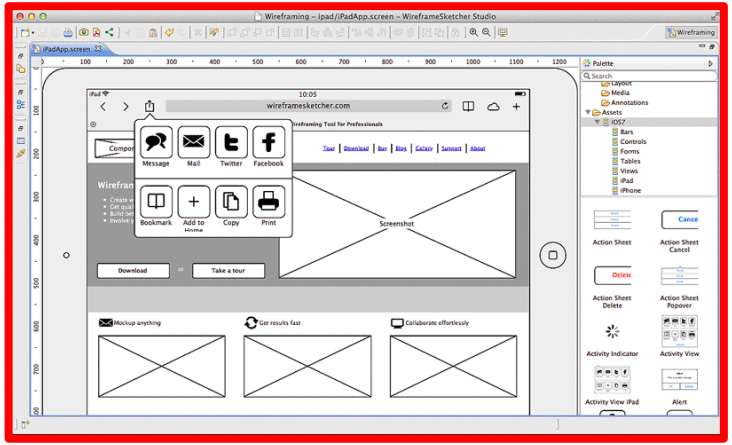


A “Little More” Detail

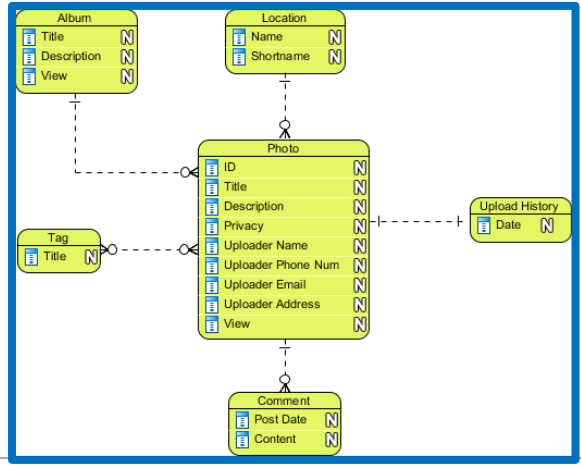


(Some) Application Design Methodologies

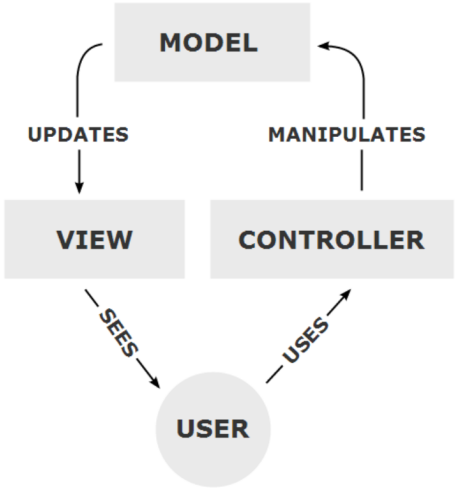
- Start with User Interface
  - Roles and personas.
  - Wireframes for pages and interactions.
  - Page flow/transition diagrams.

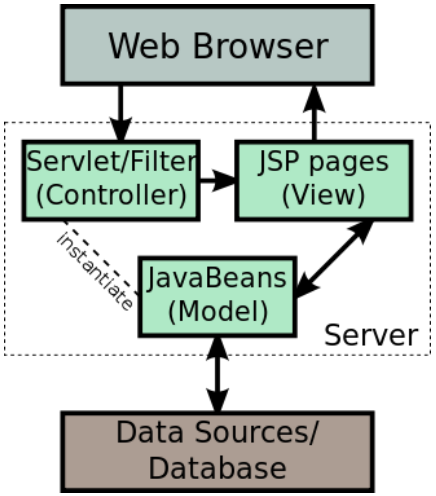


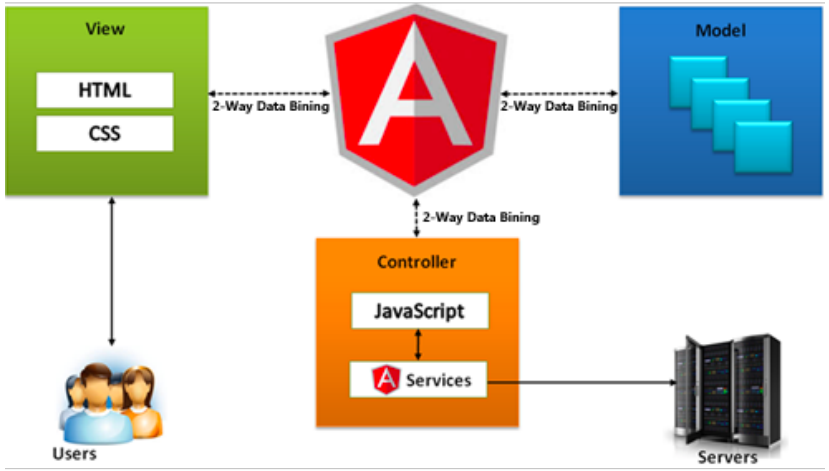
- Start with Data Model
  - Entity types, properties.
  - Relationships/associations.
  - Constraints.
  - Operations.



Model – View – Controller







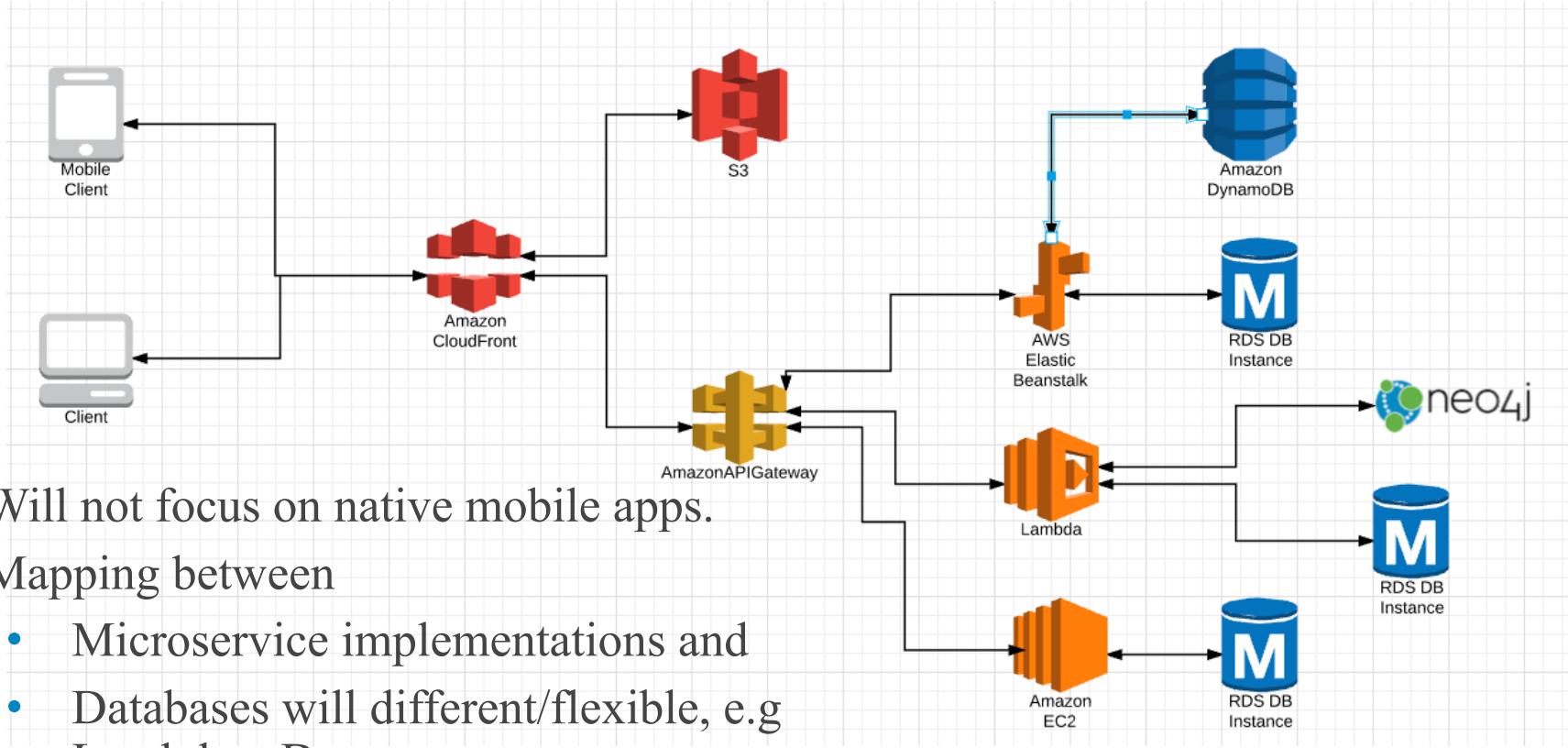
Model – View – Controller is a common paradigm

- 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.
- Out 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

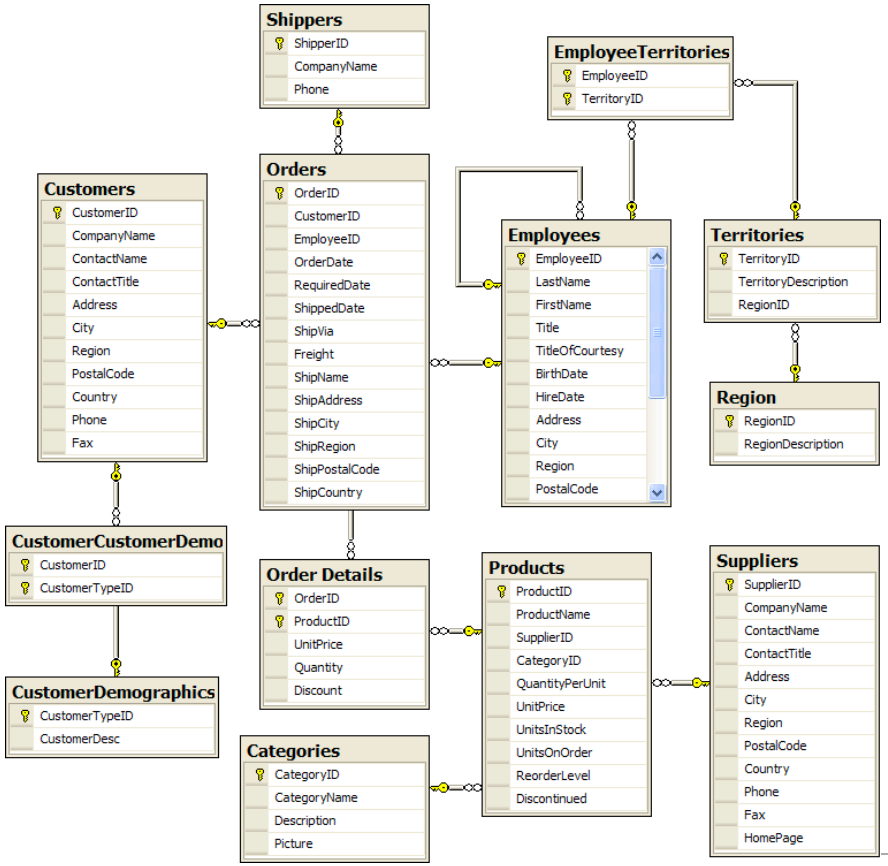


- Will not focus on native mobile apps.
- Mapping between
  - Microservice implementations and
  - Databases will different/flexible, e.g Lambda – Dynamo.

Let's Design an Application

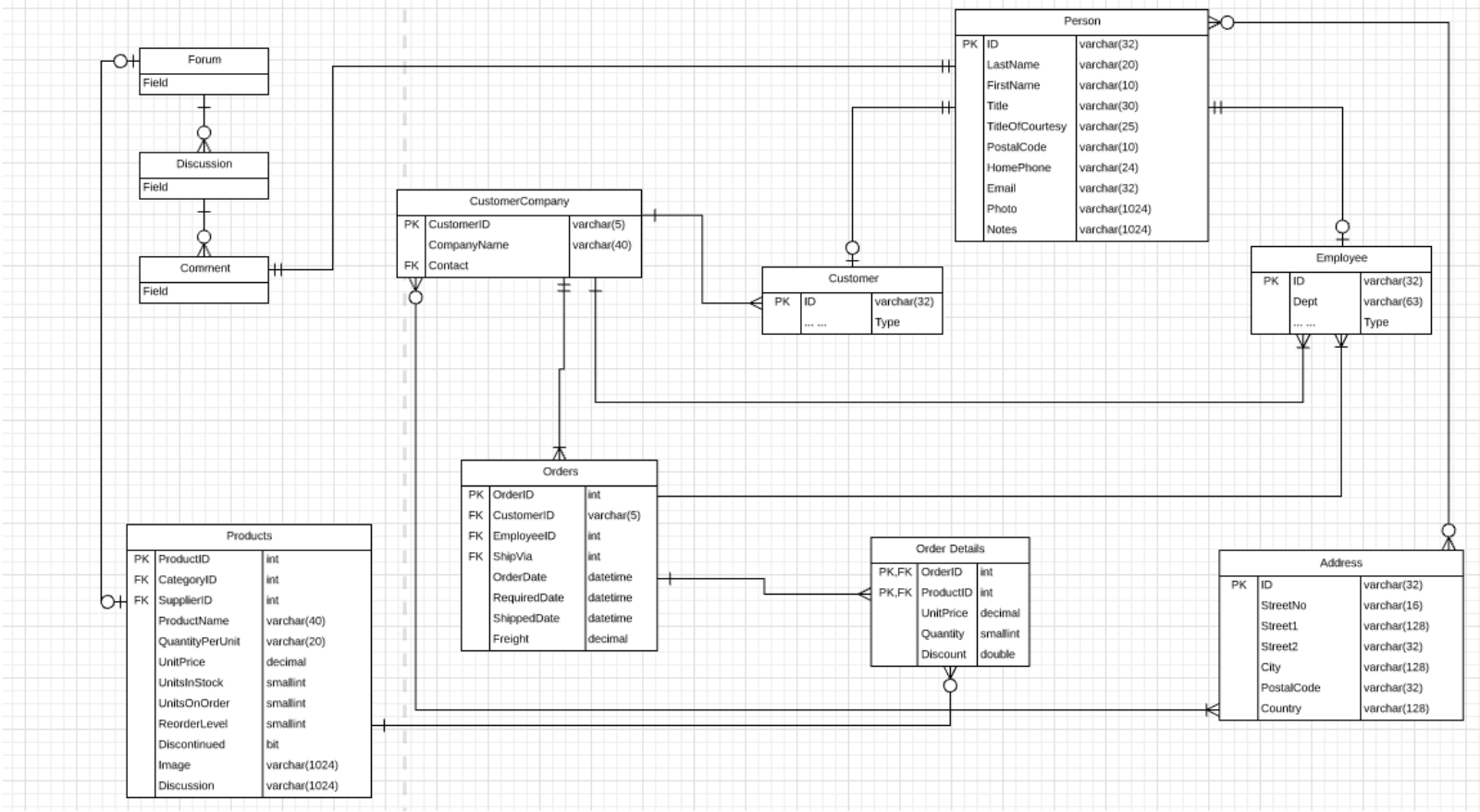
Datamodel

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



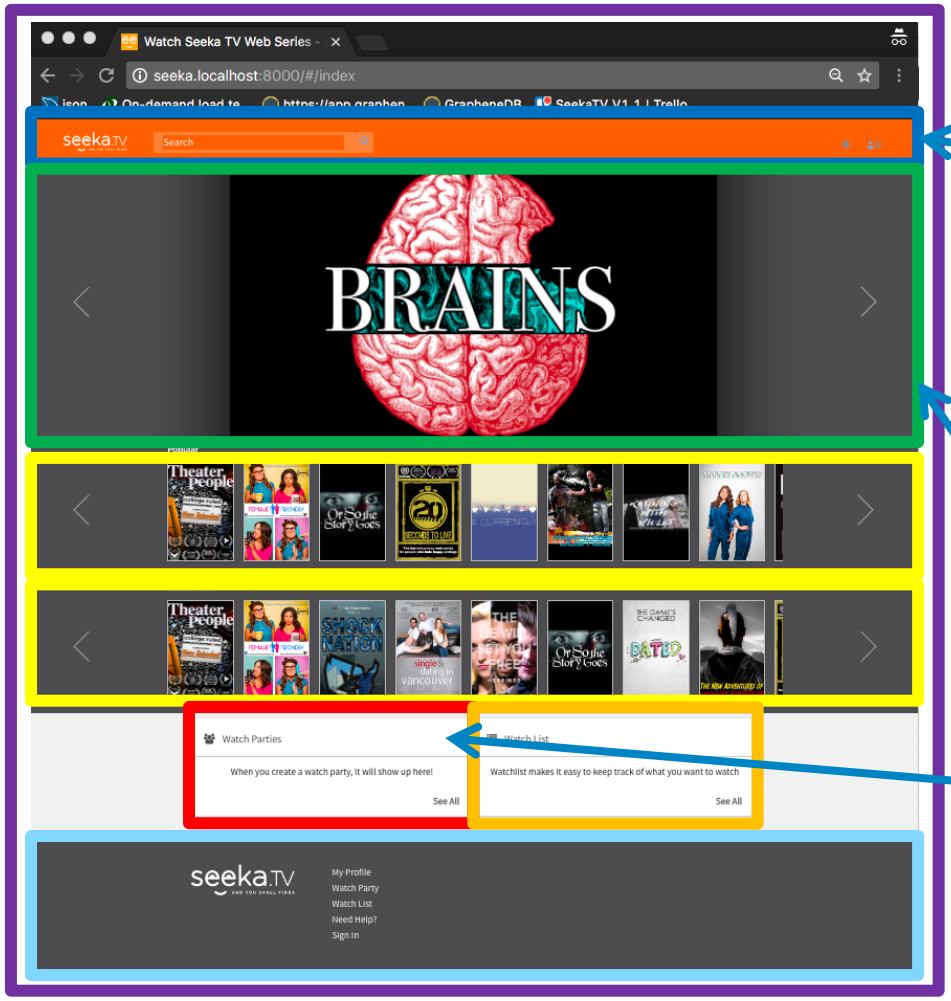
Motivation

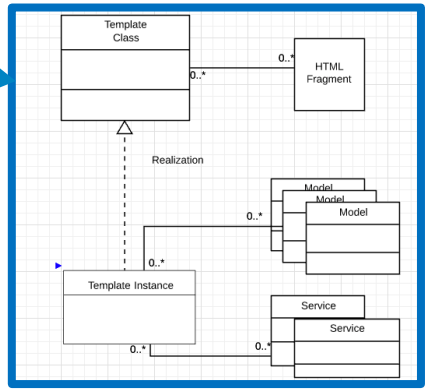
This is not a database or database modeling course. So, why are we implementing this model? It provides a foundation for some interesting use cases:

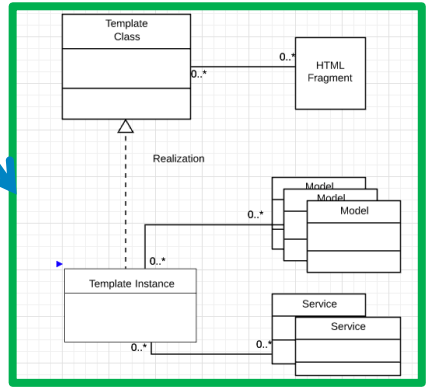
- Model of a resource with properties in a DB and links to web resources, e.g.
  - Photos of people and products.
  - Product discussion forums.
- Designing a good REST API, e.g.
  - /orders/213/orderdetails/456/product
  - /orders?q='CustomerID=123&shipped=false'
  - /orders/123/OrderDetails?f='ProductID,Quantity'
  - 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

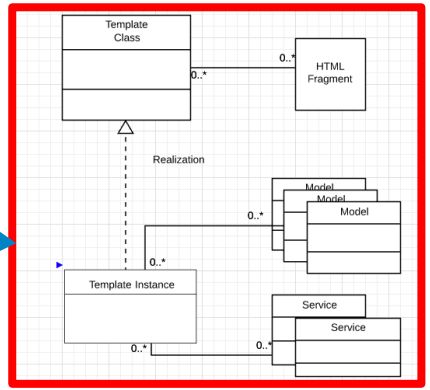
User Interface

UI
Conceptual Implementation

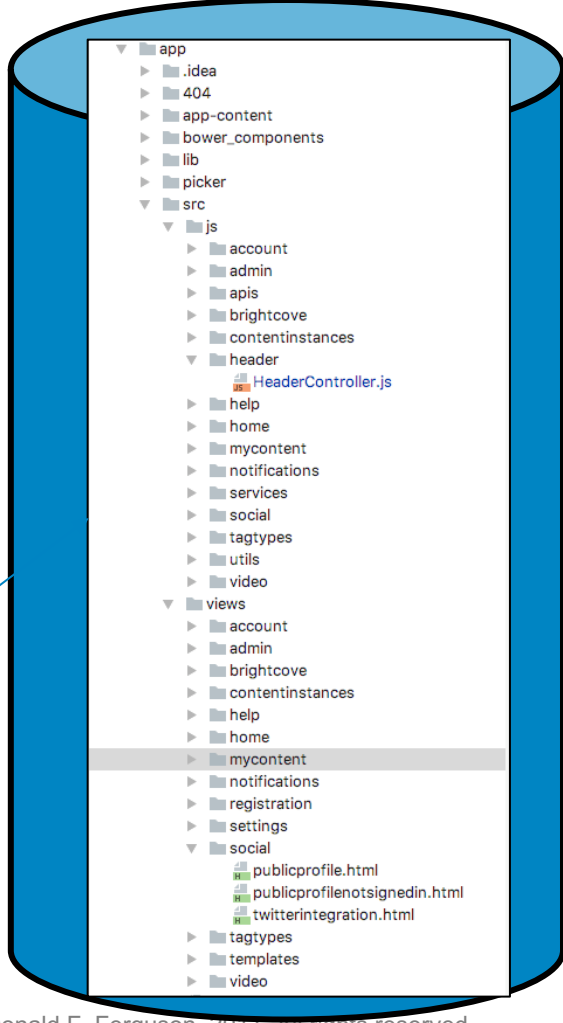








Demo Web Content Structure



Local File System

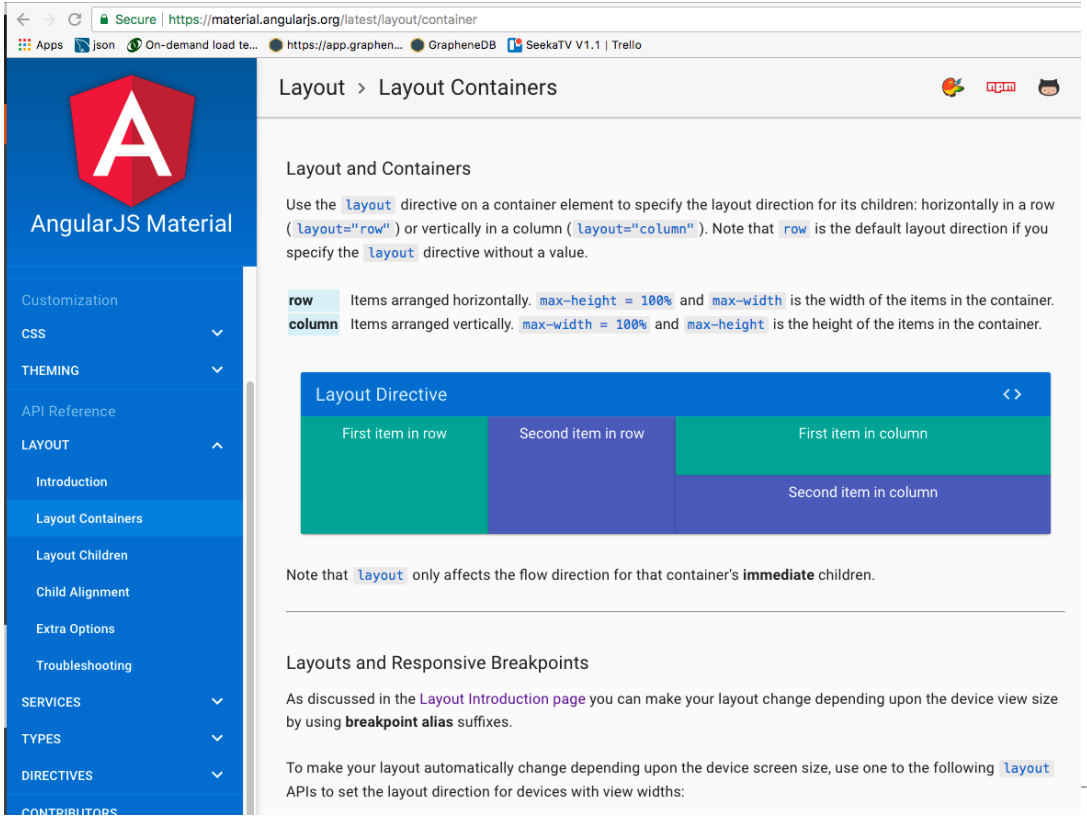
app — python -m SimpleHTTPServer — 135×24

```
KeyboardInterrupt
dyn-209-2-218-24:app donaldferguson$ clear

dyn-209-2-218-24:app donaldferguson$
dyn-209-2-218-24:app donaldferguson$
dyn-209-2-218-24:app donaldferguson$ pwd
/Users/donaldferguson/git/sparq-connexion/app
dyn-209-2-218-24:app donaldferguson$ python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
127.0.0.1 - - [11/Sep/2017 08:57:08] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [11/Sep/2017 08:57:08] "GET /app-content/lib/css/bootstrap.css?version=buildNumber HTTP/1.1" 200 -
127.0.0.1 - - [11/Sep/2017 08:57:08] "GET /app-content/fonts/roboto.css?version=buildNumber HTTP/1.1" 200 -
127.0.0.1 - - [11/Sep/2017 08:57:08] "GET /app-content/fonts/exo-2.css?version=buildNumber HTTP/1.1" 200 -
127.0.0.1 - - [11/Sep/2017 08:57:08] "GET /app-content/fonts/material-icons.css?version=buildNumber HTTP/1.1" 200 -
127.0.0.1 - - [11/Sep/2017 08:57:08] "GET /app-content/fonts/font-awesome.css?version=buildNumber HTTP/1.1" 200 -
127.0.0.1 - - [11/Sep/2017 08:57:08] "GET /app-content/css/main.css?version=buildNumber HTTP/1.1" 200 -
127.0.0.1 - - [11/Sep/2017 08:57:08] "GET /lib/mdetect.js HTTP/1.1" 200 -
127.0.0.1 - - [11/Sep/2017 08:57:08] "GET /lib/jquery.min.js HTTP/1.1" 200 -
127.0.0.1 - - [11/Sep/2017 08:57:08] "GET /lib/angular.min.js HTTP/1.1" 200 -
127.0.0.1 - - [11/Sep/2017 08:57:08] "GET /lib/angular-animate.min.js HTTP/1.1" 200 -
127.0.0.1 - - [11/Sep/2017 08:57:08] "GET /lib/angular-aria.min.js HTTP/1.1" 200 -
127.0.0.1 - - [11/Sep/2017 08:57:08] "GET /lib/angular-ui-router.min.js HTTP/1.1" 200 -
127.0.0.1 - - [11/Sep/2017 08:57:08] "GET /lib/angular-sanitize.min.js HTTP/1.1" 200 -
127.0.0.1 - - [11/Sep/2017 08:57:08] "GET /lib/angular-material.min.js HTTP/1.1" 200 -
```

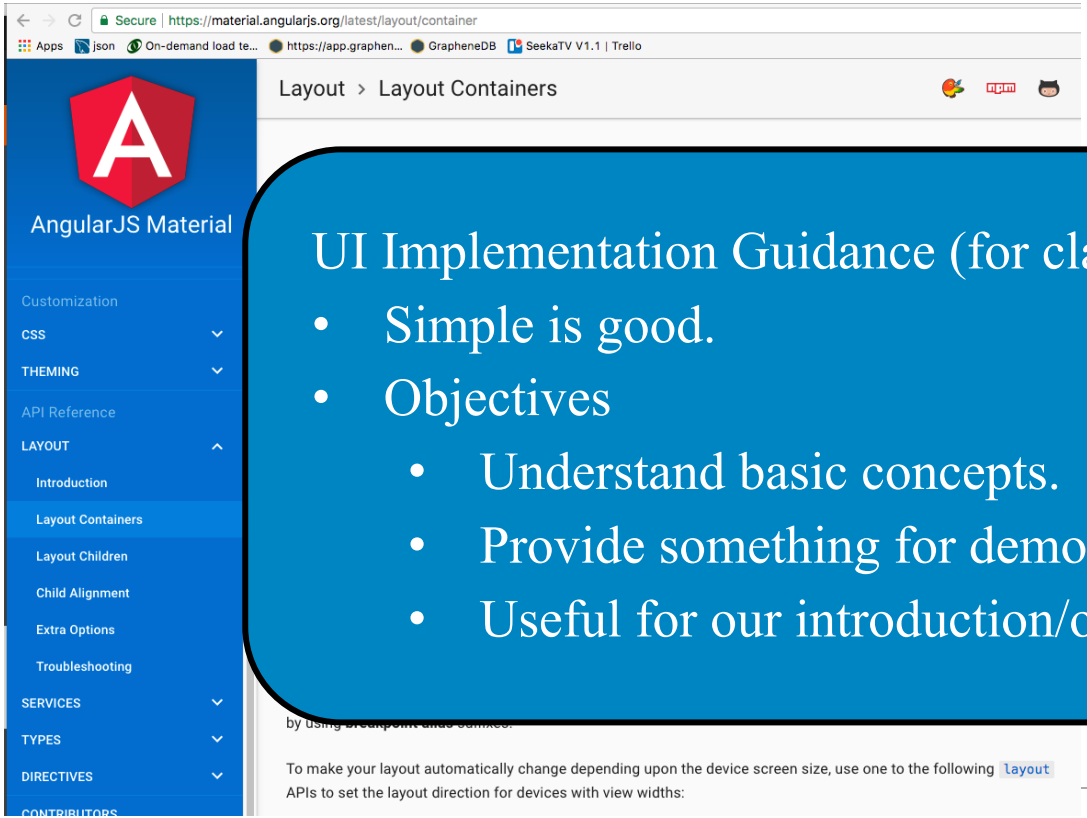
Local Web Server

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



- 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/)



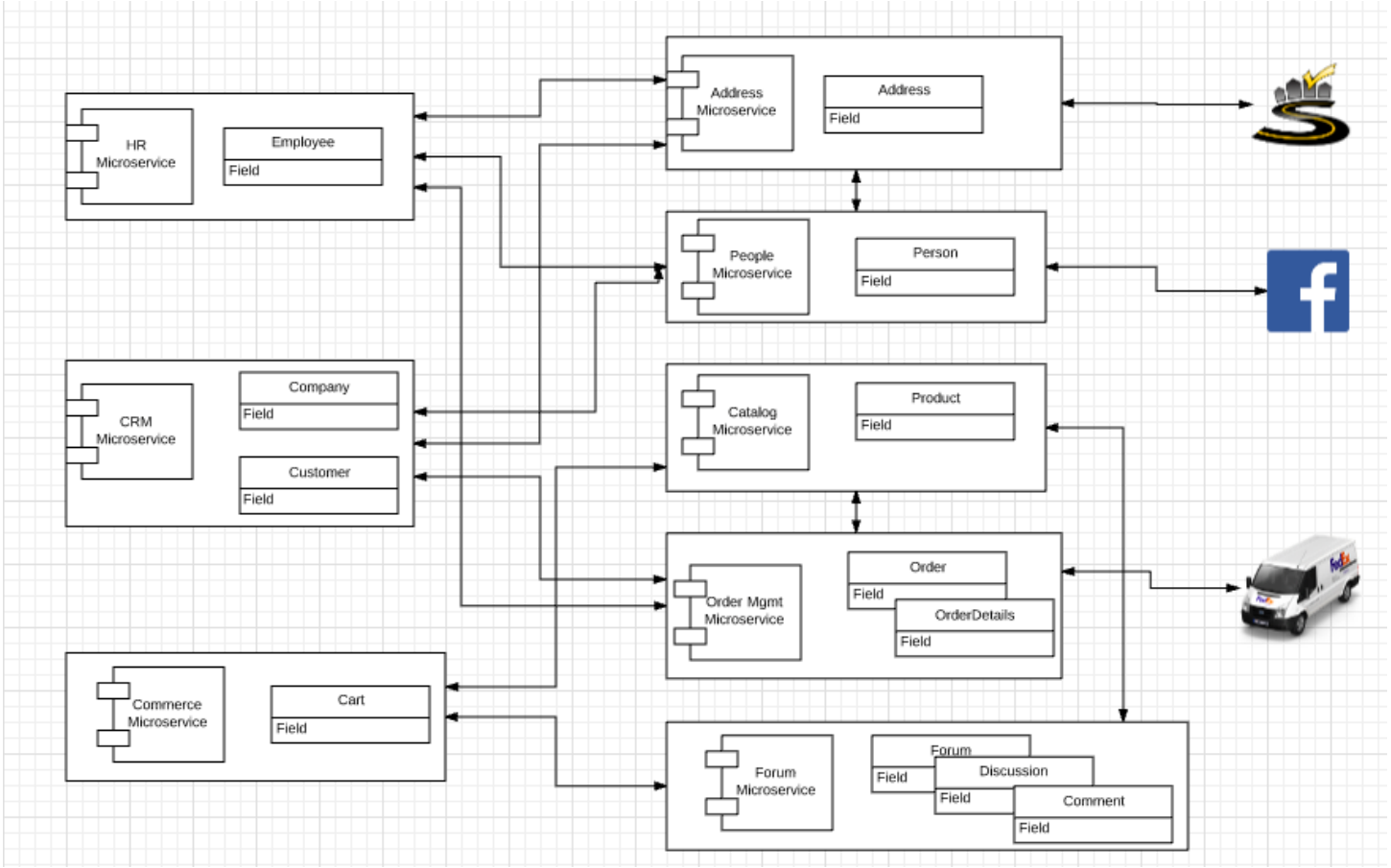
- Class examples and lectures will use Angular JS Material.

UI Implementation Guidance (for class)

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

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)*

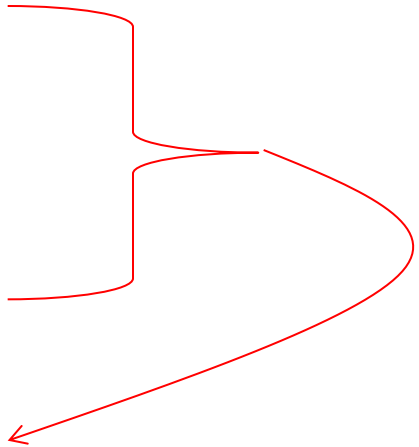
- Collection/Set: /customers
  - POST: Create a new customer
  - GET: Return all customers, and implement
    - Query
    - Pagination
- Specific Instance: /customers/dff9 (dff9 is a unique, primary key)
  - GET: Some or all fields.
  - PUT: Update some or all fields.
  - DELETE: Delete the specific 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) 202 - Resource accepted (Operation Pending)
UPDATE	PUT	202 - Accepted (Operation Pending) 204 - Success (Operation Complete)
DELETE	DELETE	202 - Accepted (Operation Pending) 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
Unprecessable 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 *resou*

```
{
  "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-ba 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 li

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"
    },
}
```

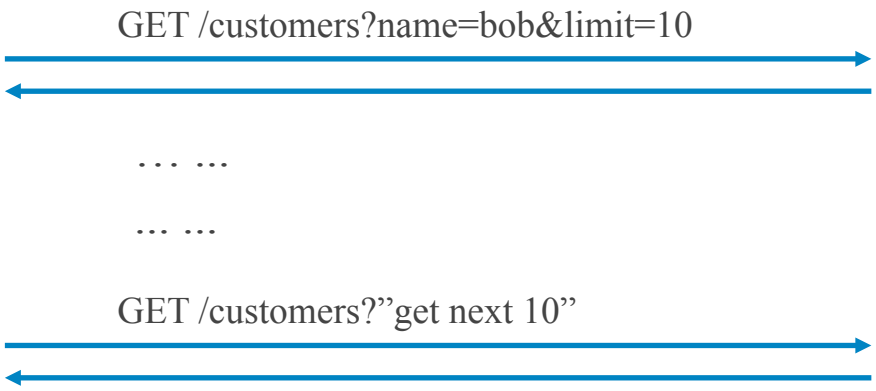
Data fields followed by links section.

```
{
    …
    "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.

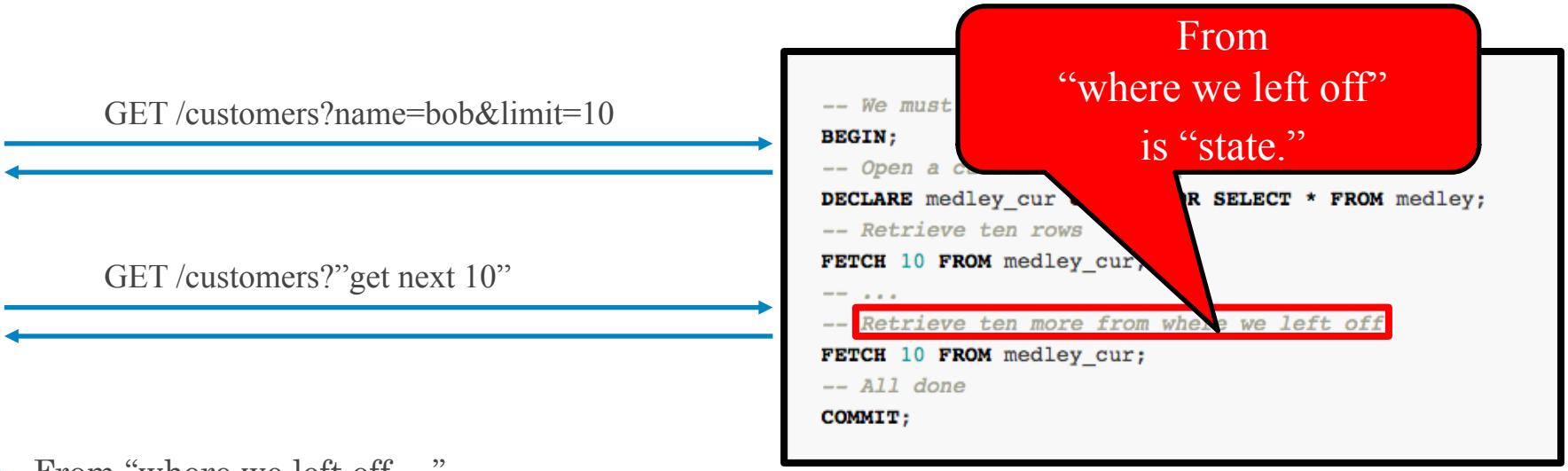
Traditional Solution is Iterator/Cursor




```
-- 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

http://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_Query.html

Example

Next Request

```
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_WIT
H)
            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 LastEvaluatedKe
y)
        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
});
```

Previous Response

```
        }
    }
  ],
  "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
}
```

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

returns.

{“data”:
[{“user_id”:”42”, “name”:”Bob”,
“links”:[{“rel”:”self”, “href”:”http://
{“user_id”:”22”, “name”:”Frank”,
“links”: [{“rel”:”self”, “href”:”http://api.exa
{“user_id”:”125”, “name”: “Sally”,
“links”:[{“rel”:”self”, “href”:”http://api.ex

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.

“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”}]}

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

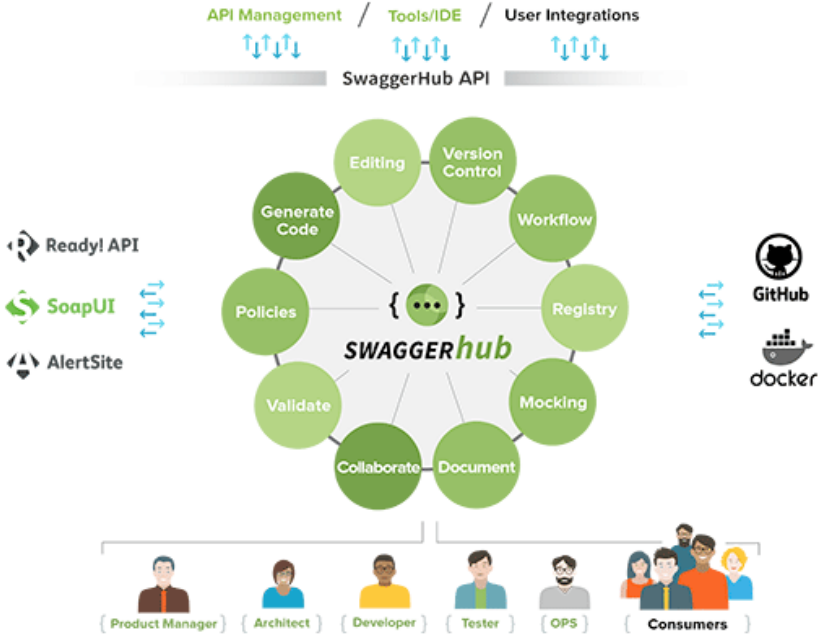
Design • Document • Discover

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

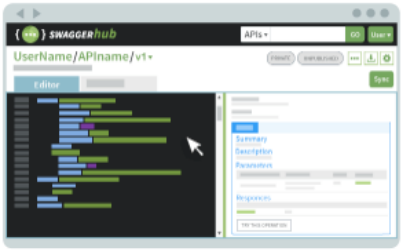
- Collaborative design.
- Interactive documentation.
- And an easily searchable registry of Swagger-based APIs.

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

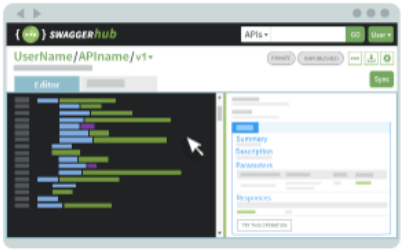


Use our intuitive editors to create your API definition and collaborate with others on their API definitions. On-the-fly validation keeps you honest.

Code Gen

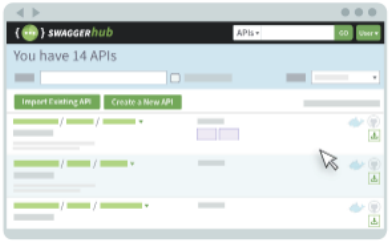
Get your development project off to a quick start by using our client and server code templates.

Versioning



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

API Registry

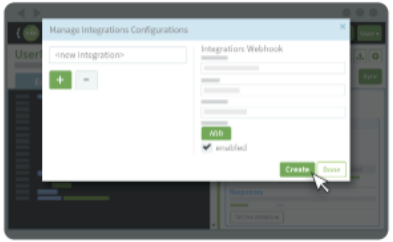


Browse our list of Swagger-based APIs and explore them using our interactive documentation.

Domains

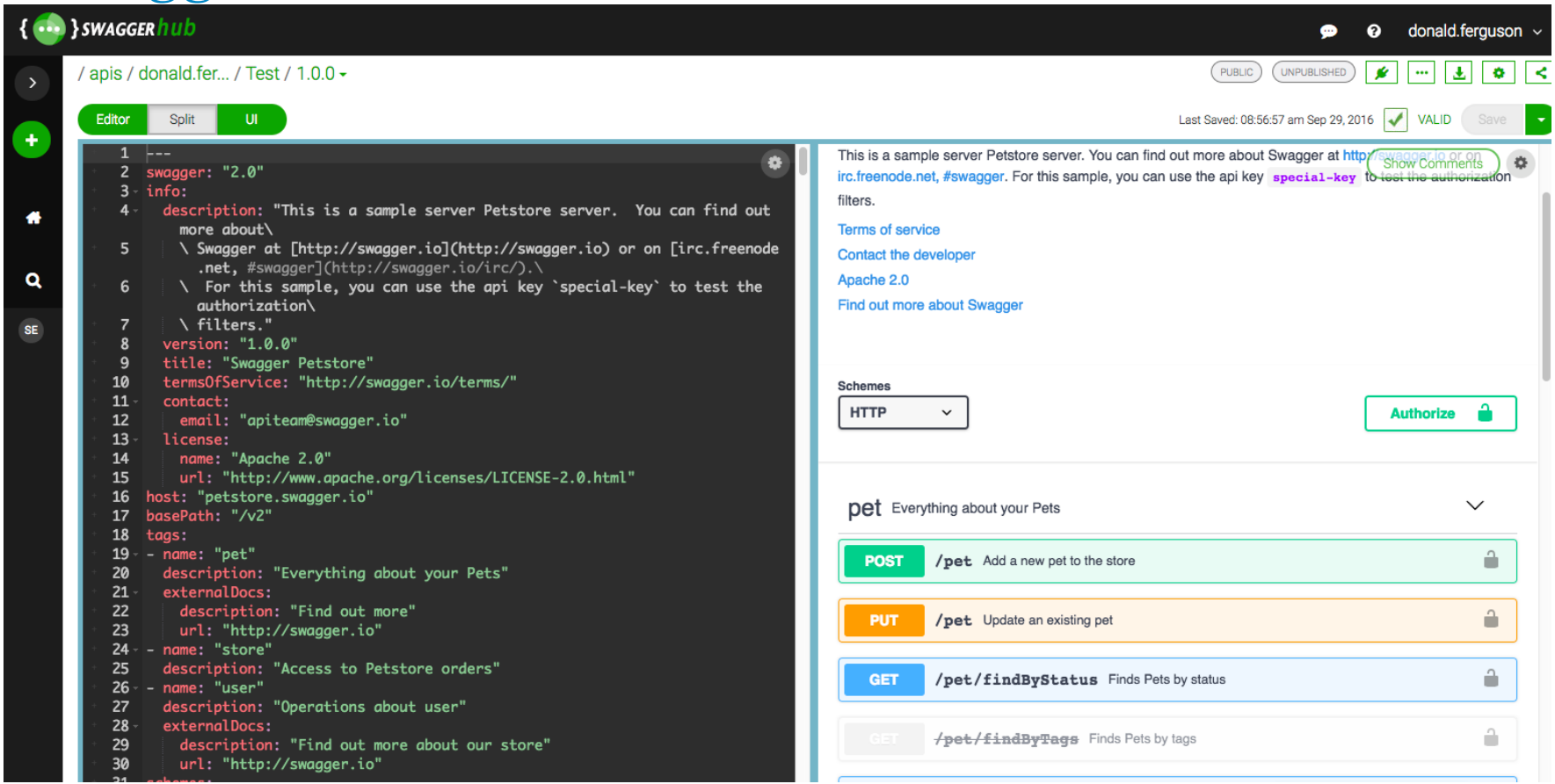
Store all your re-usable components that can later be used across multiple API definitions, saving you time

Plugins

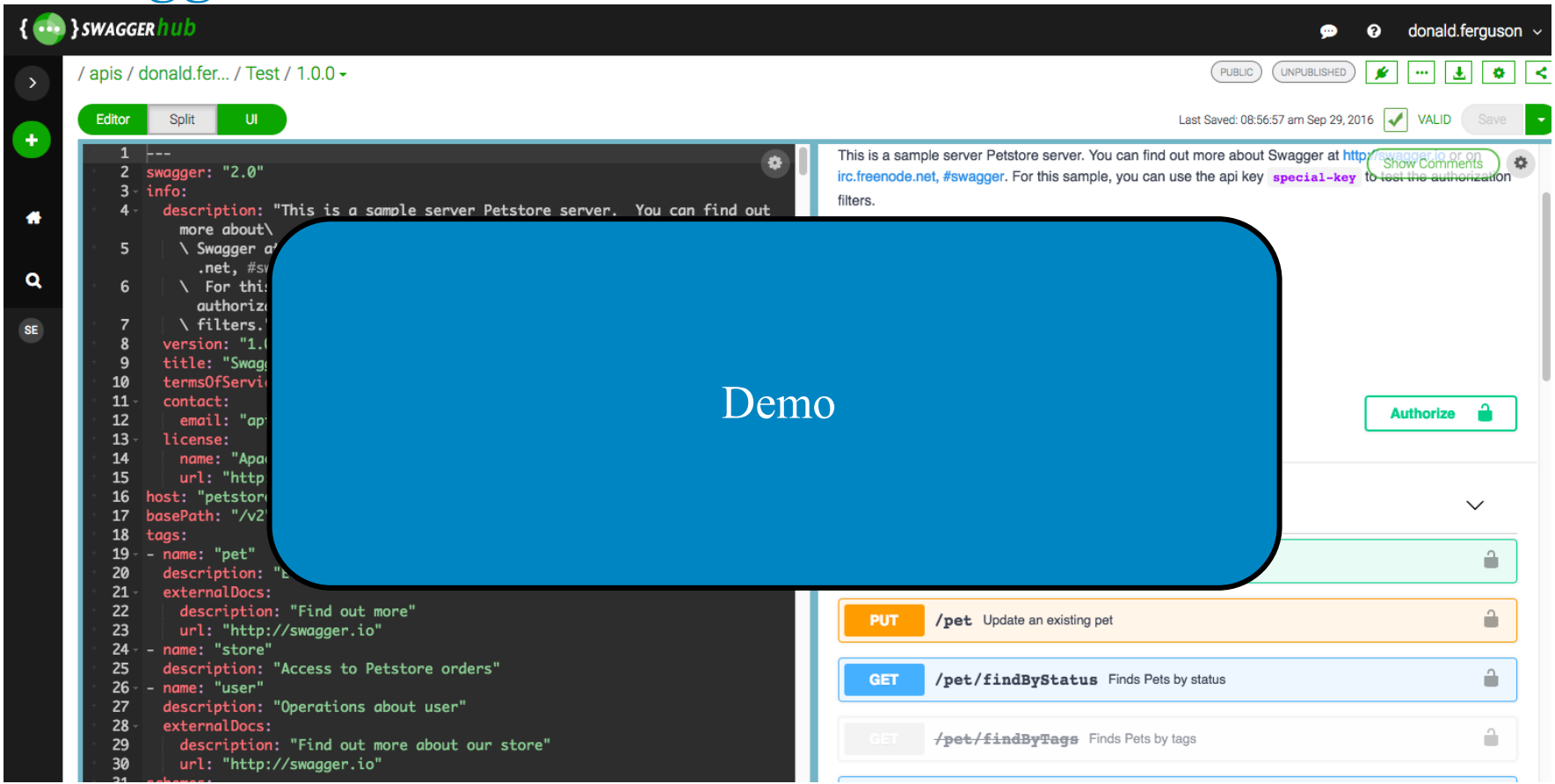


LEARN MORE

Swagger Editor



Swagger Editor



1st 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

1st Project – Part 1

- 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.

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