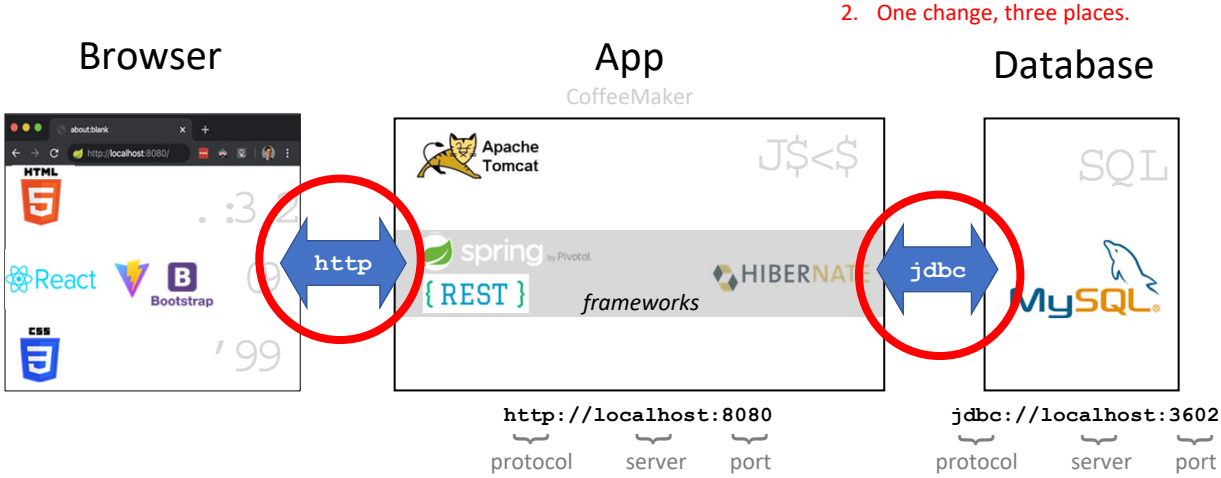
REST APIs

CSC 326

1

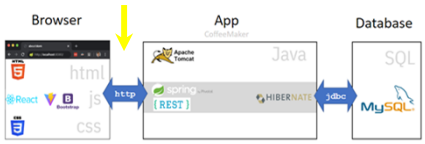
Key Pain Points

Transfer Representation of Data
1. Slow.
2. One change, three places.



2

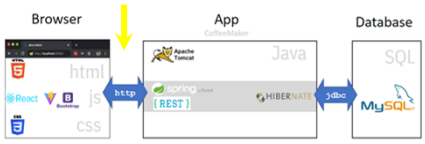
What are API's?



- Problem: *How can my software talk to other software?*
- Solution: ***APIs*** *Application Programming Interface*
- Example: { REST } • **REST** = **RE**presentational **S**tate **T**ransfer

3

REST APIs



- Provide an interface for clients (typically web systems) to interact with back end models, controllers, and services
- Consume and produce JSON
- Actions use HTTP requests, like GET, POST, DELETE, PUT

GET	Retrieve record(s)	GET /recipes/	Retrieves all recipes
DELETE	Delete record(s)	DELETE /recipes/1	Deletes recipe with id of 1
POST	Create record	POST /recipes/	Creates a new recipe
PUT	Update record	PUT /recipes/3	Updates recipe 3

This is like the CRUD operations from the Backend lecture!

You'll need to send along some extra JSON data for the last two

4

Pragmatic REST API Design

- Audience: developer
  - Goal: Make developer as successful as possible
- Nouns = Resources (top level object)
  - Collections
  - Element of collection
- Verbs = Operations on collections and element
  - Use HTTP Verbs
    - POST (**c**reate)
    - GET (**r**ead)
    - PUT (**u**pdate)
    - DELETE (**d**elete)

`/dogs`

`/dogs/1234`

Web API Design

5

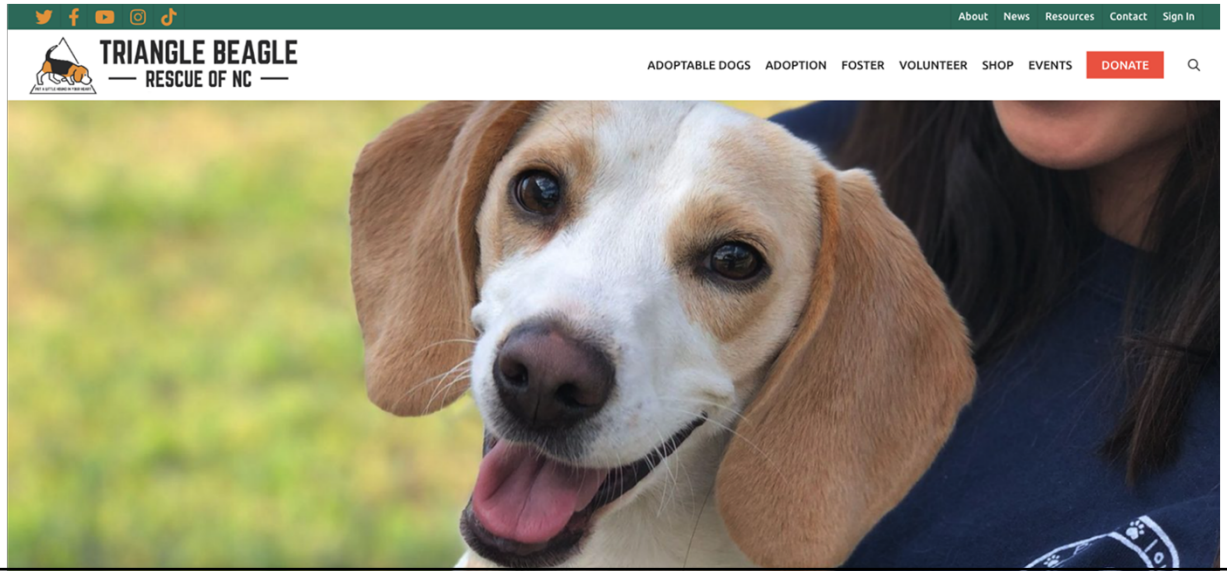
Example REST API Interactions

Resource	POST (create)	GET (read)	PUT (update)	DELETE (delete)
/dogs	Create a new dog	List dogs	Bulk update dogs	Delete all dogs
/dogs/1234	Error	Show Bo	If exists, update Bo If not, error	Delete Bo

Web API Design - Table from Page 7

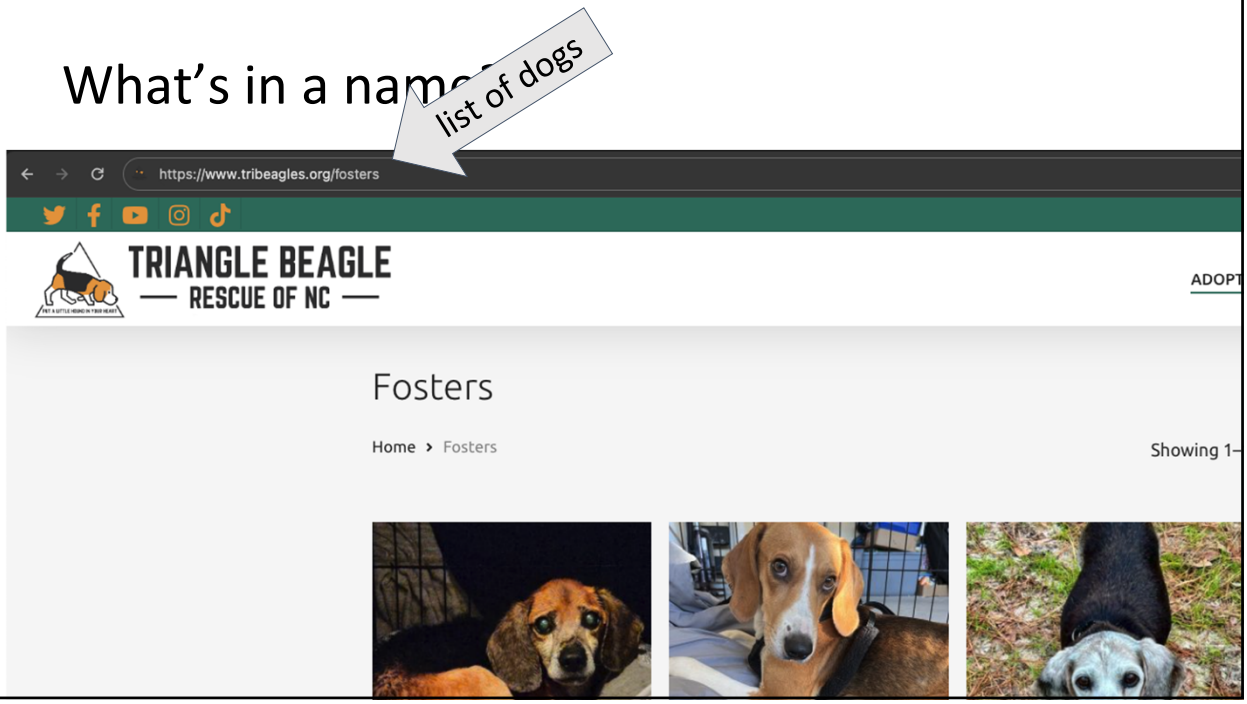
6

Example: Dog Rescue Website



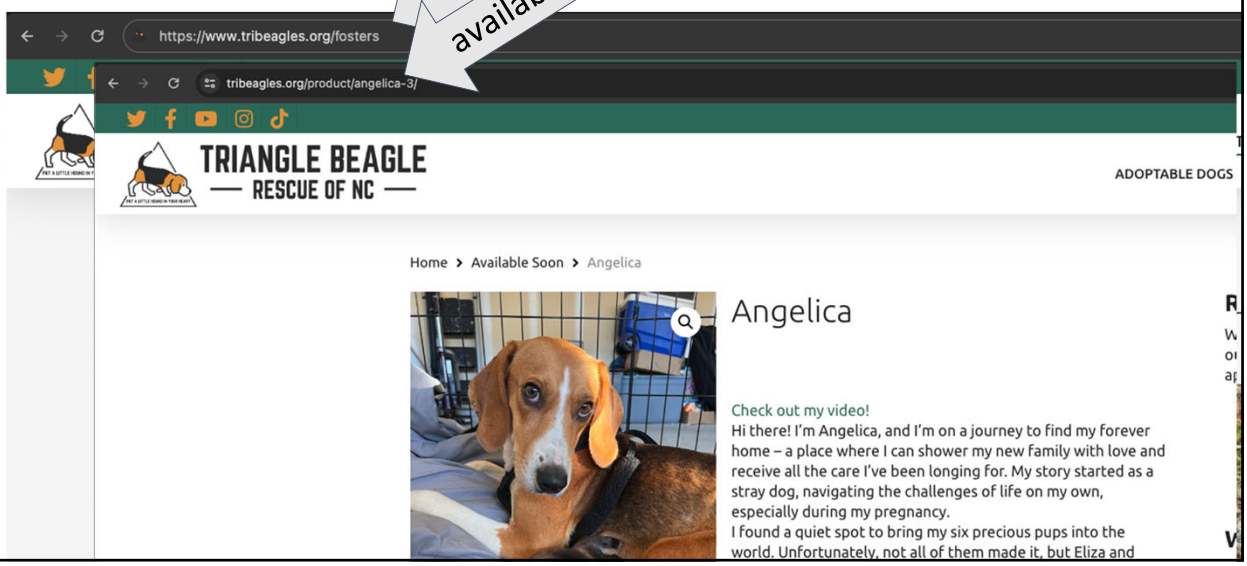
7

What's in a name?



8

What's in a name?



9

What's in a name?



10

Resource Associations

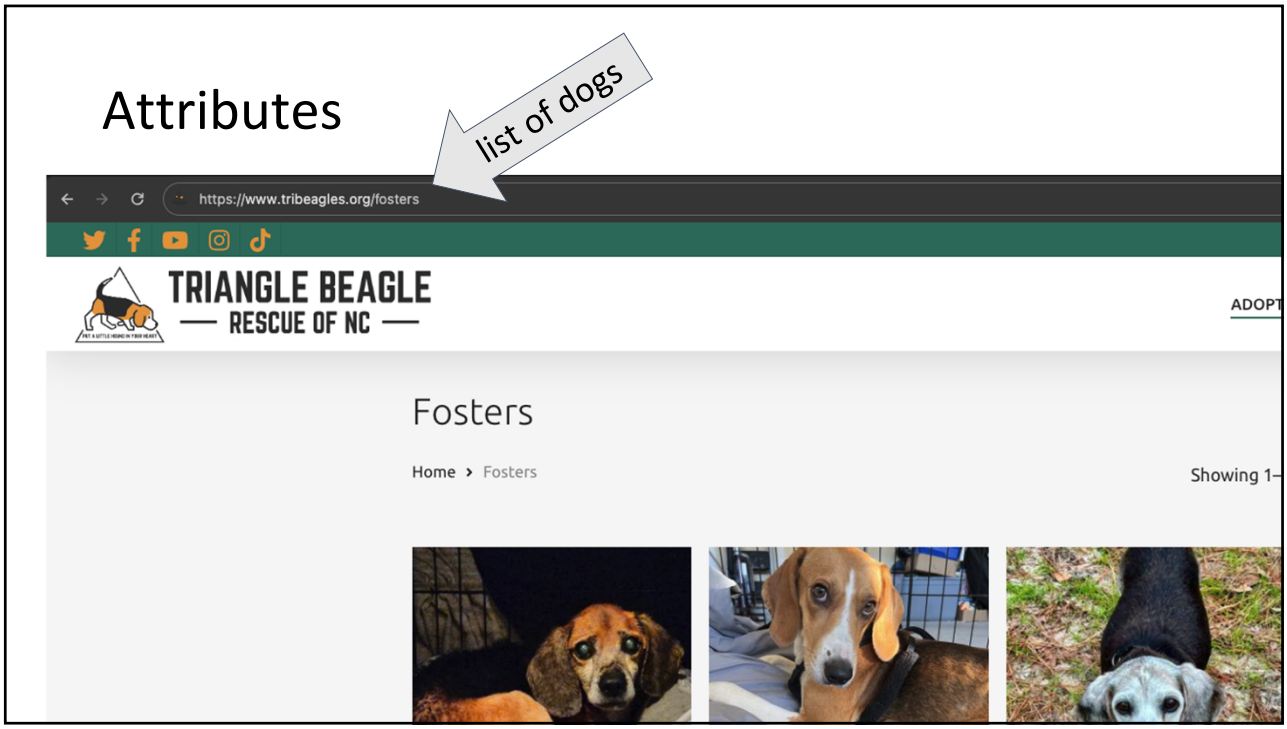
- Handling associations adds another level to the URL
  - Once have primary key for one level (an element) you don't need to include levels above it
  - Pattern `/resource/identifier/resource`
- Example: Dogs have owners
  - Get all the dogs for an owner `GET /owners/5678/dogs`
  - Create a new dog for an owner `POST /owners/5678/dogs`

Parameters: States and Attributes

- States and resource attributes shouldn't be handled as associations
  - Add them behind the HTTP question mark

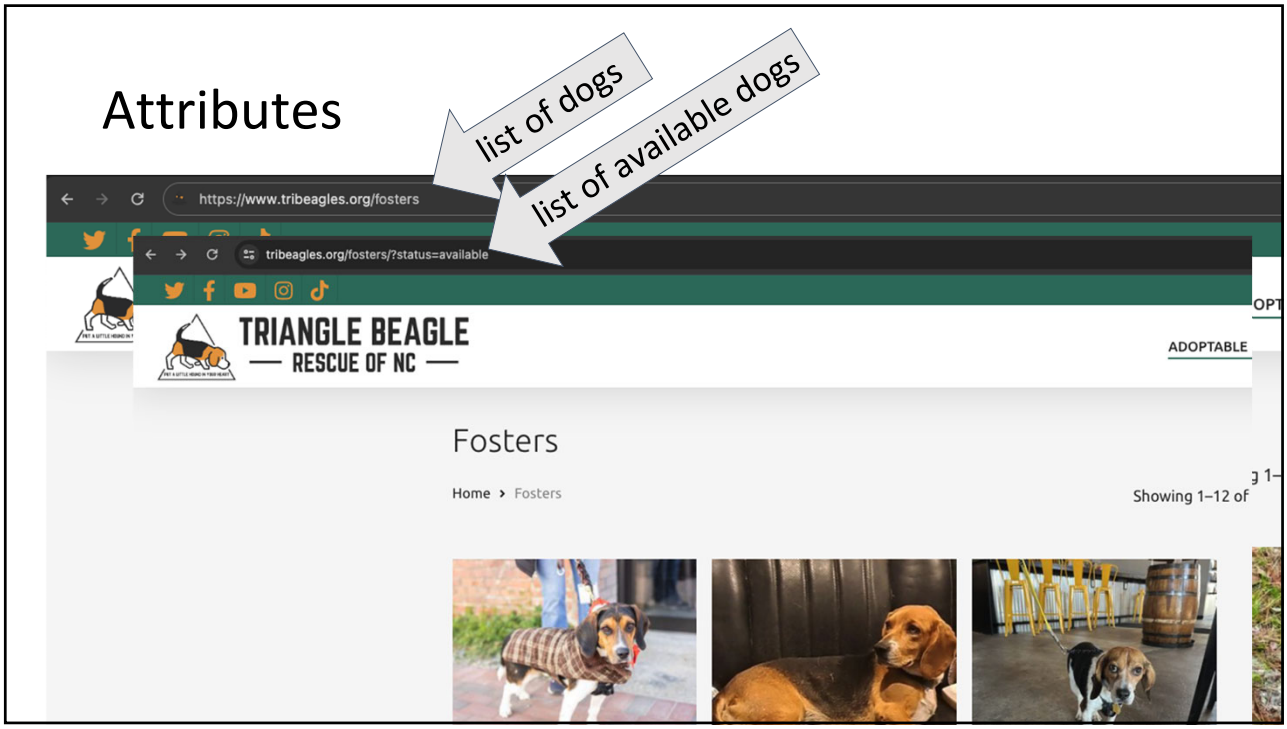
```
GET /dogs?color=red&state=running&location=park
```

Attributes



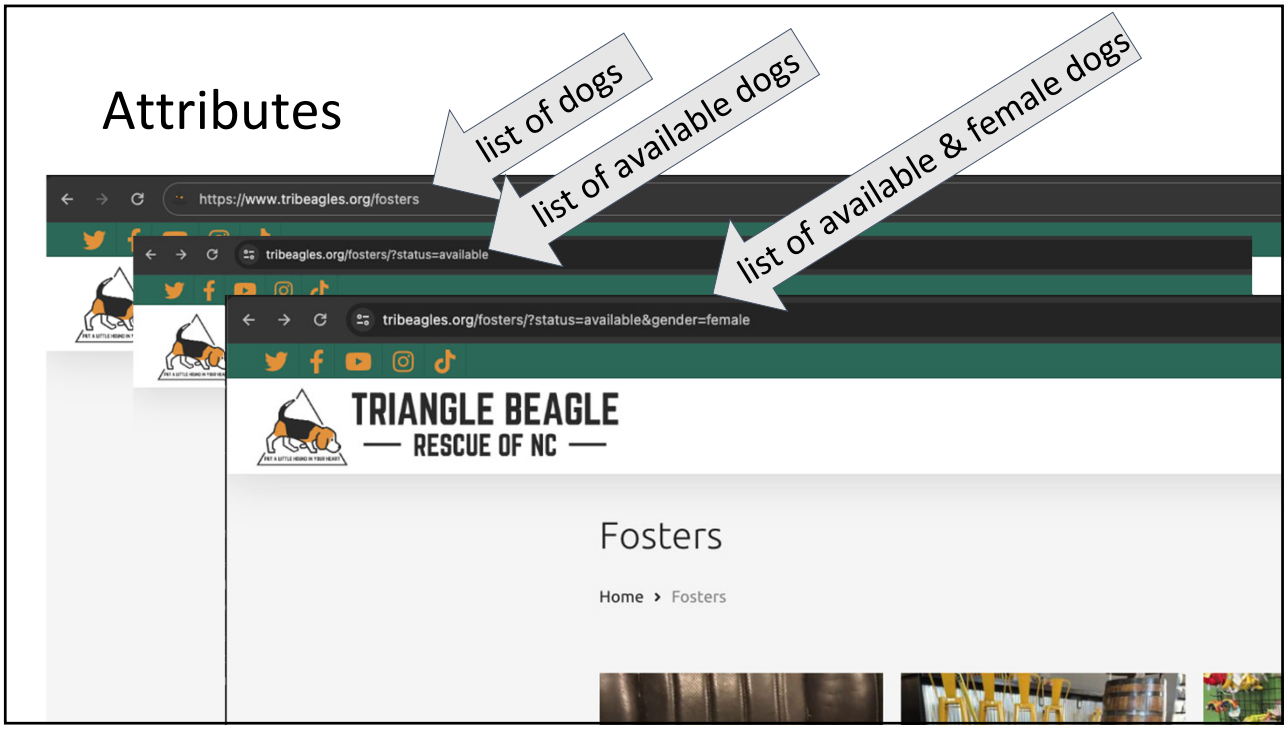
13

Attributes



14

Attributes



15

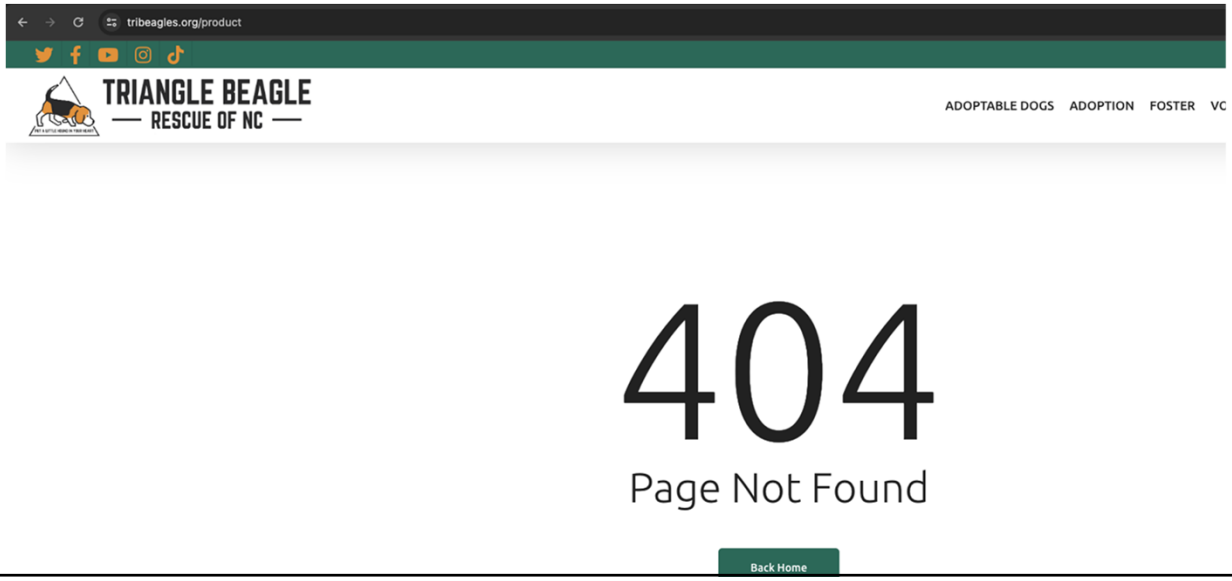
Handling Errors

- Errors provide context to client misuse of the black-box behind the API
  - Use HTTP status codes
    - Everything worked – success
      - 200 - OK
    - The application did something wrong – client error
      - 400 – Bad Request
    - The API did something wrong – server error
      - 500 – Internal Server Error
    - Add additional codes as appropriate
  - Returned messages should be verbose

Web API Design

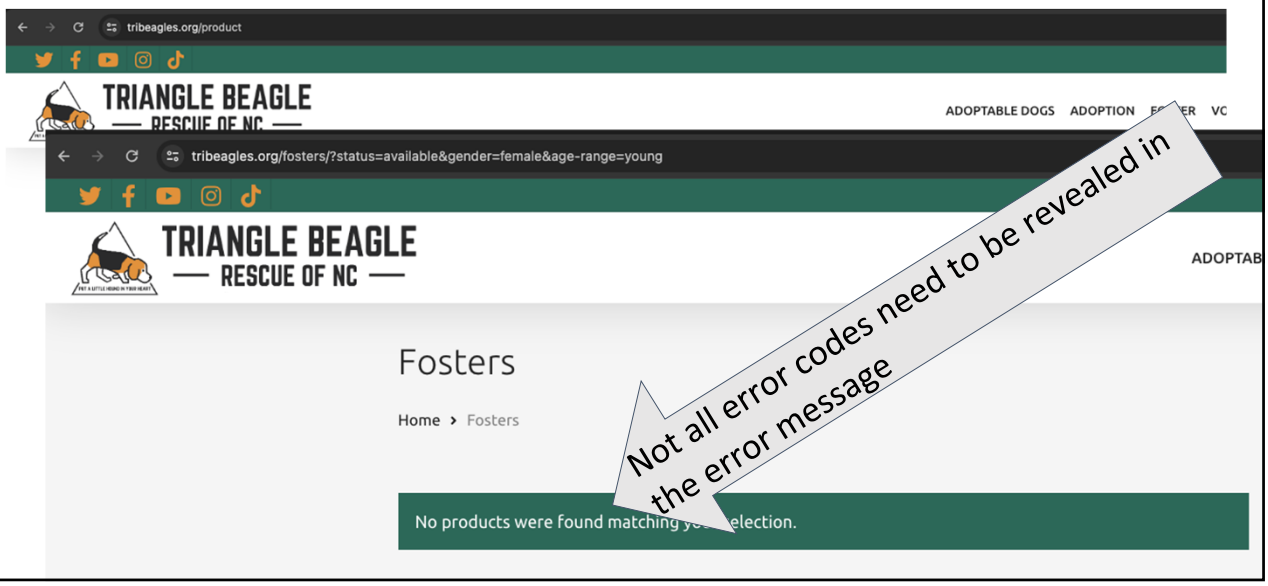
16

Errors



17

Errors

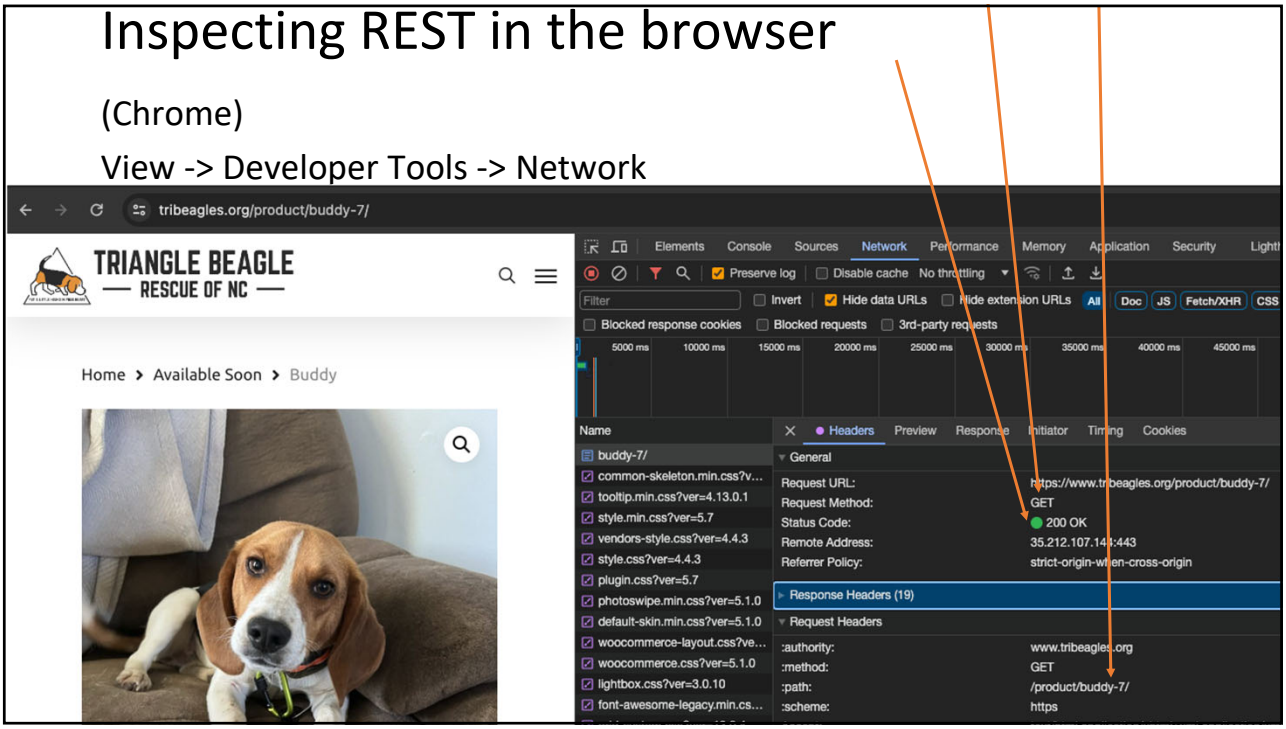


18

Inspecting REST in the browser

(Chrome)

View -> Developer Tools -> Network



19

Exercise: REST APIs

- Provide the REST API interactions for the following scenarios
  - Create a new recipe
  - Get the coffee recipe
  - Get all the recipes

20

GET recipe (M2 instructions)

GET Routes

GET routes are used to *fetch* data from the API, and as such, we care about seeing a description of the API endpoint and the data that is *returned* from the API.

GET Example

Name of function: `getRecipe(String name)`

Route: `GET /recipes/{name}`

Brief Description: This API endpoint will return a recipe with the name provided, or an error (404/Not Found) if a recipe with the provided name cannot be found.

Data Details:

If we have a recipe by the name `Mocha`, we would call our endpoint as `GET /recipes/Mocha` and expect the API to return a 200/OK and JSON data that looks like:

```
{  // start of Recipe object
  'name': 'Mocha',
  'price': 10,
  'sugar': 2,
  'milk': 3,
  'coffee': 5,
  'chocolate': 3
}
```

If we call our endpoint instead as `GET /recipes/nope` we would expect to get a 404/Not Found and no recipe JSON data.

21

RESTful APIs

- Architectural Properties
  - Performance – appropriate component interactions
  - Scalability – support large numbers of components and interactions among components
  - Simplicity of interfaces
  - Modifiability of components
  - Visibility of communication between components by service agents
  - Portability of components by moving program code with the data
  - Reliability when failures within components, connectors, or data

Wikipedia

22

RESTful APIs

- Architectural Constraints
  - Client-server
  - Stateless
  - Cacheable
  - Layered system
  - Code on demand (optional)
  - Uniform Interface
    - Identification of resources
    - Manipulation of resources through these representations
    - Self-descriptive messages
    - Hypermedia as the engine of application state

Wikipedia and REST API Tutorial

23

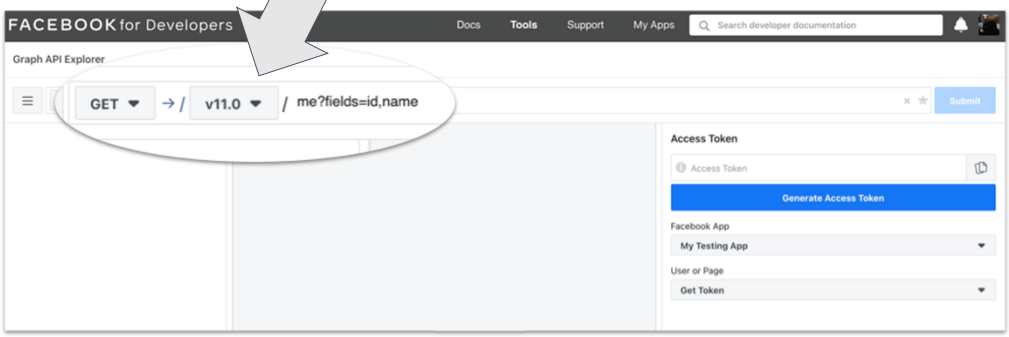
Other Considerations

- Versioning – APIs should be versioned
  - Make version mandatory in URL/header
- Pagination
  - Provide way to limit number of responses
  - Offset and limit
- Partial Response
  - Let clients request a subset of information about a resource
- Authentication – Use common libraries (OAuth)

Web API Design

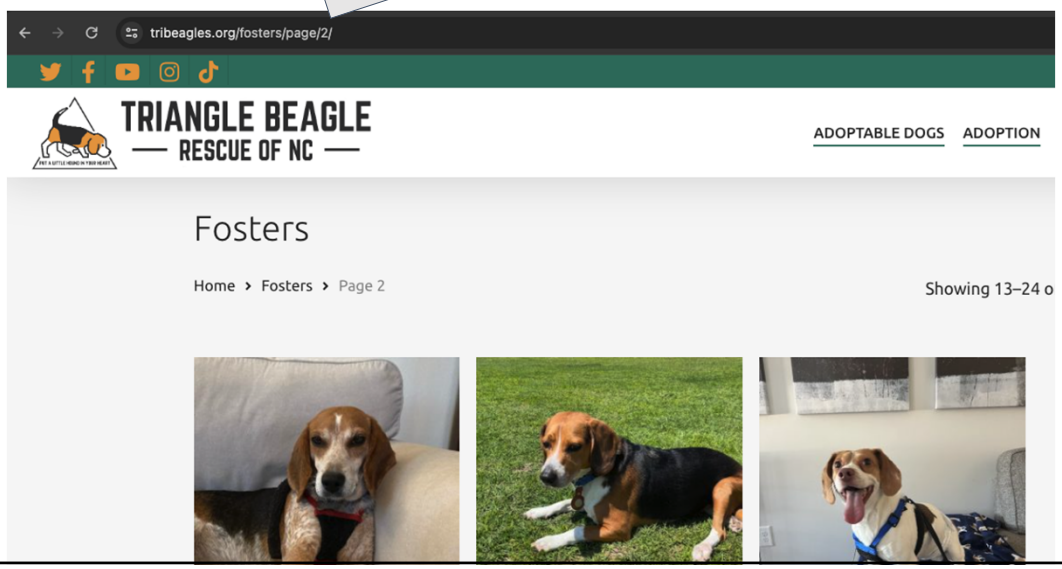
24

Versioning



25

Pagination



26

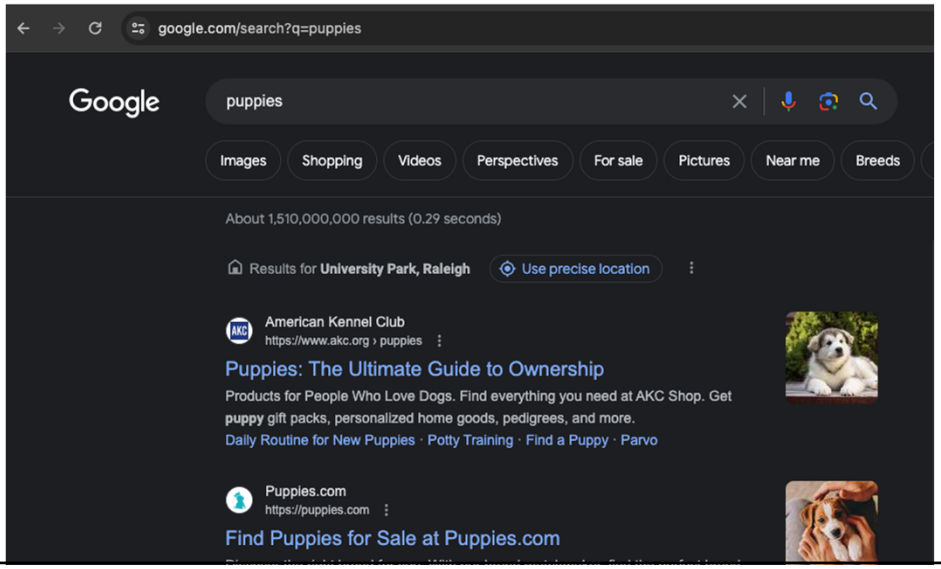
Responses without Resources

- Making sure you aren't having your client do things the API could do
  - Common actions `/convert?from=EUR&to=CNY&amount=100`
- Use verbs
  - Clearly define actions vs. resources in documentation
  - Consistency matters
  - Search
    - Global `/search?q=fluffy+fur`
    - Scoped `/owners/5678/dogs?q=fluffy+fur`

Web API Design

27

Global search example



28

Response Formats

- XML (Extensible Markup Language)
  - Store and transport data
  - Schema defines the tags that describe the data
  - Libraries to help with binding XML to objects (JAXB)
- JSON (JavaScript Object Notation)
  - Lightweight data-interchange format
  - Collection of name/value pairs (object)
  - Ordered list of values (array)

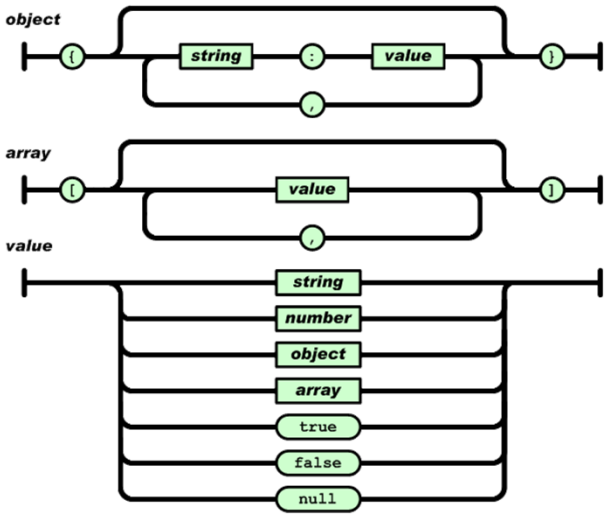
```
var myObject = JSON.parse(response);
```

W3Schools – XML and JSON.org

29

JSON Structures

- Object
- Array
- Value



JSON.org

30

GET recipe (M2 instructions)

Start of JSON Object

string:value,
string:value,
string:value,
…

GET Routes

GET routes are used to *fetch* data from the API, and as such, we care about seeing a description of the API endpoint and the data that is *returned* from the API.

GET Example

Name of function: `getRecipe(String name)`

Route: `GET /recipes/{name}`

Brief Description: This API endpoint will return a recipe with the name provided, or an error (404/Not Found) if a recipe with the provided name cannot be found.

Data Details:

If we have a recipe by the name `Mocha`, we would call our endpoint as `GET /recipes/Mocha` and expect the API to return a 200/OK and JSON data that looks like:

```
{  // start of Recipe object
  'name': 'Mocha',
  'price': 10,
  'sugar': 2,
  'milk': 3,
  'coffee': 5,
  'chocolate': 3
}
```

If we call our endpoint instead as `GET /recipes/nope` we would expect to get a 404/Not Found and no recipe JSON data.

31

API Design

- Joshua Bloch – How to Design a Good API and Why it Matters
  https://www.youtube.com/watch?v=aAb7hSCtvGw#t=2733

32

Example APIs

- Facebook Graph API
  - https://developers.facebook.com/docs/graph-api
- Twitter
  - https://dev.twitter.com/rest/public
- GitHub
  - https://docs.github.com/rest
- Salesforce – Force API
  - https://developer.salesforce.com/page/REST_API
- Twilio
  - https://www.twilio.com/docs/api/rest

33

API General Principles (1)

- Do one thing well
  - You can never remove
  - If it's hard to name, that's a bad sign (more later)
- Implementation should not impact API
  - Don't "leak" implementation details
  - Façade pattern
- Minimize accessibility of EVERYTHING
  - Information hiding
  - Modules used independently

Joshua Block Slides

34

API General Principles (2)

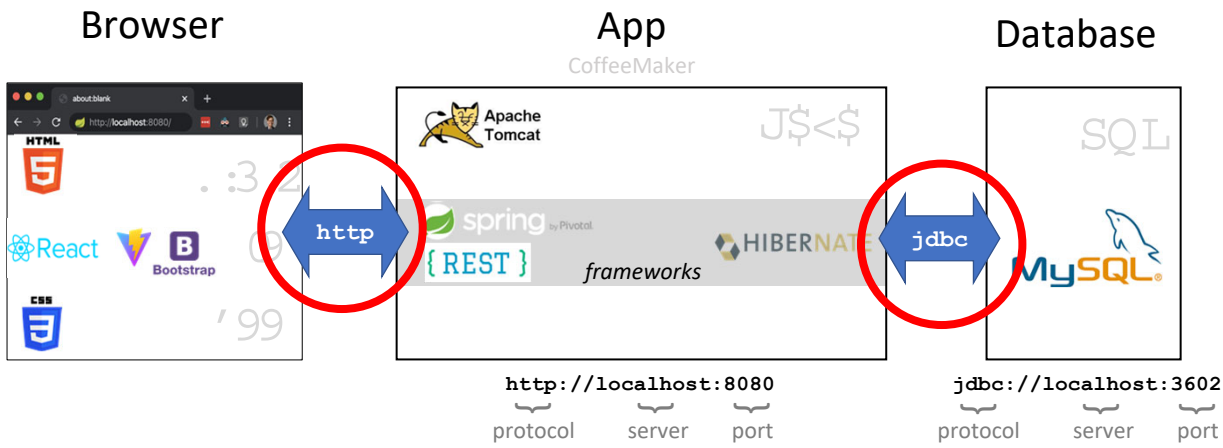
- Names Matter
  - Names should be self-explanatory
  - Consistent naming – words mean the same
- Don't make the client do something the API could *easily* do
  - Reduce need for boilerplate code

Joshua Block Slides

35

Key Pain Points

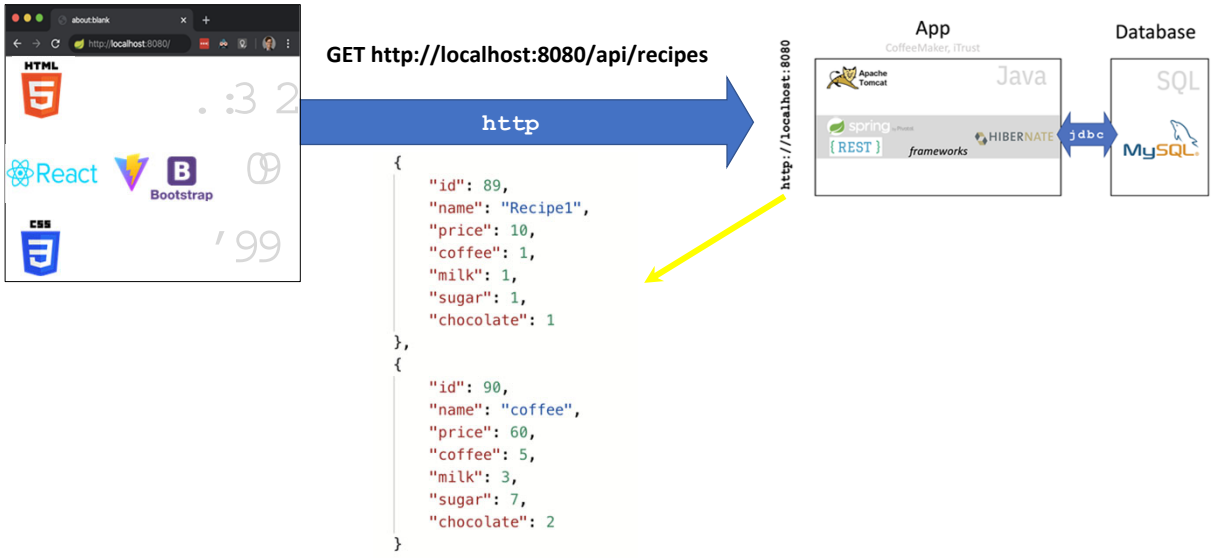
Transfer Representation of Data
1. Slow.
2. One change, three places.



36

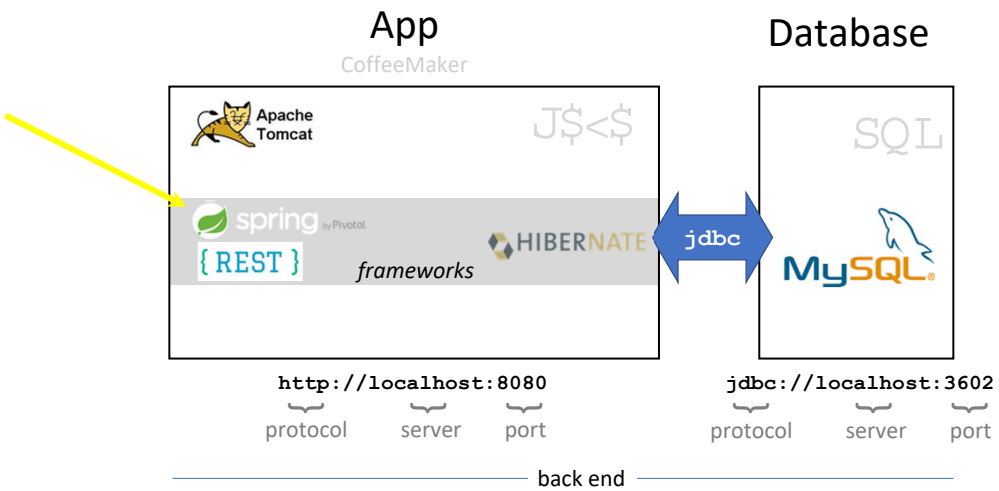
REST returns JSON

Browser



37

326 App Architecture



38

{ REST } APIs in spring by Pivotal



Spring provides libraries to simplify the creation of REST APIs

```
@CrossOrigin("*")
@RestController
@RequestMapping("/api/recipes")
public class RecipeController {
    @Autowired
    private RecipeService recipeService;
    @GetMapping
    public List<Recipe> getRecipes() {
        return service.findAll();
    }
}
```

Identifies a REST controller in Spring.

REST API path

Connection to the *Service object

Type of REST request the method responds to.

39

{ REST } APIs in spring by Pivotal



Spring provides libraries to simplify the creation of REST APIs

GET mapping

```
@GetMapping("{name}" )
public ResponseEntity<RecipeDto> getRecipe
                                    (@PathVariable("name") String name)
{
    RecipeDto recipeDto = recipeService.findByName(name);
    return ResponseEntity.ok(recipeDto);
}
```

Path variable with name "name"

Annotation connects to path variable {name}

Method returns a ResponseEntity
- Body of generic type – recipe if success, error otherwise
- HTTP status – use constants in HttpStatus - .ok method returns 200.

40

{ REST } APIs in spring by Pivotal



Spring provides libraries to simplify the creation of REST APIs

POST mapping

Body of the POST request as object

```
@PostMapping
public ResponseEntity createRecipe ( @RequestBody RecipeDto recipeDto) {
   if (recipeService.isDuplicateName(recipeDto.getName())) {
      return new ResponseEntity<>(recipeDto, HttpStatus.CONFLICT);
   }
   if (recipeService.getAllRecipes().size() < 3) {
      RecipeDto savedRecipeDto = recipeService.createRecipe(recipeDto);
      return ResponseEntity.ok(savedRecipeDto);
   }
   return new ResponseEntity<>(recipeDto, HttpStatus.INSUFFICIENT_STORAGE);
}
```

Method returns a ResponseEntity
- Body of generic type – string success message
- HTTP status – use constants in HttpStatus

41

REST API Design Documentation

Name of function

Function: **`getRecipe(String name)`**

Route: **`GET /recipes/{name}`**

REST API endpoint (relative URL)

Description

Description: This API endpoint will return a recipe with the name provided, or an error (404/Not Found) if a recipe with the provided name cannot be found.

Data Details:
If we have a recipe by the name Mocha, we would call our endpoint as **`GET /recipes/Mocha.`** The API should return a 200/OK and JSON data that looks like:

Data details

```
{ //start of Recipe object
    'name': 'Mocha',
    'price': 10,
    'sugar': 2,
    'milk': 3,
    'coffee': 5,
    'chocolate': 3
}
```

If we call our endpoint instead as **`GET /recipes/nope`** we would expect to get a 404/Not Found and no recipe JSON data.

42

References

- R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” Ph.D. Dissertation, University of California, Irvine, 2000.
  https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
- T. Fredrich, “Learn REST: A RESTful Tutorial,” http://www.restapitutorial.com/
- B. Mulloy, “Web API Design,” Apigee. http://apigee.com/about/resources/ebooks/web-api-design
- C. Parnin, “Frameworks, APIs, and Technology Stacks” Lecture Notes.
- Wikipedia
  - “Representation State Transfer”. Last modified 10/27/2015 at 23:06. https://en.wikipedia.org/wiki/Representational_state_transfer
  - “Dependency Injection”. Last modified 10/22/2015 at 12:39. https://en.wikipedia.org/wiki/Dependency_injection