RAWDATA SECTION 2

Troels Andreasen & Henrik Bulskov

HTTP

RESOURCE LOCATORS

- uniform resource locator
  - http://example.com/index.html
  - Structure:
    - the part before the ://, is what we call the **URL scheme**
    - everything after the :// will be specific to a particular scheme
    - E.G: example.com is the host and /index.html is the URL path
  - Port
    - http://example.com:80/index.html
  - Query string
    - http://example.com:80/index.html?name=peter
  - Fragment
    - http://example.com:80/index.html?name=peter#news

  - <scheme>://<host>:<port>/<path>?<query>#<fragment>

Verb	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	404 (Not Found), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	404 (Not Found), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	404 (Not Found), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

HTTP REQUEST METHODS

```
[method] [URL] [version]
[headers]

[body]
```

```
GET http://odetocode.com/ HTTP/1.1
Host: odetocode.com
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) Chrome/16.0.91
Safari/535.7
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
Referer: http://www.google.com/url?&q=odetocode
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

HTTP REQUEST HEADERS

HTTP RESPONSE

[version] [status] [reason]
[headers]

[body]

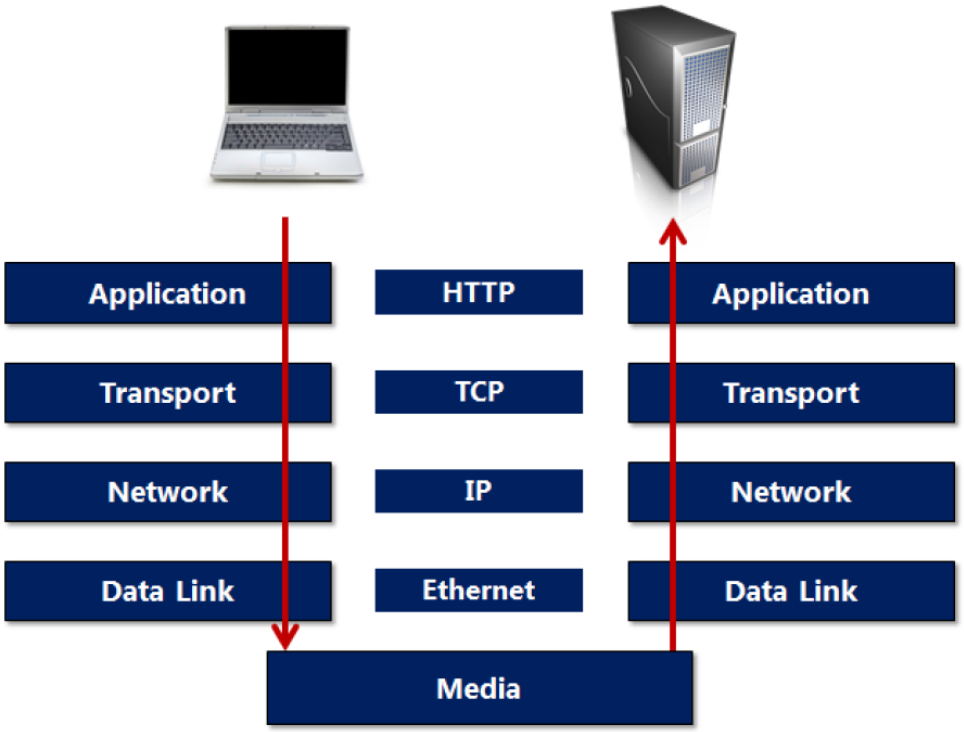
```
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/7.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Sat, 14 Jan 2012 04:00:08 GMT
Connection: close
Content-Length: 17151

<html>
<head>
    <title>.NET-related Articles, Code and Resources</title>
</head>
<body>
 ... content ...
</body>
</html>
```

RESPONSE STATUS CODES

Range	Category
100–199	Informational
200–299	Successful
300–399	Redirection
400–499	Client Error
500–599	Server Error

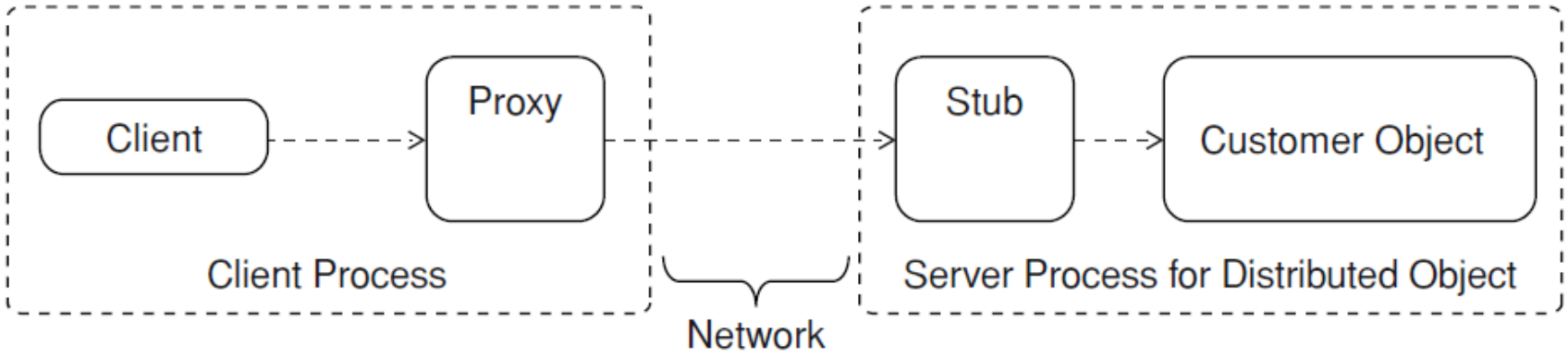
COMMUNICATION STACK



WEB SERVICES

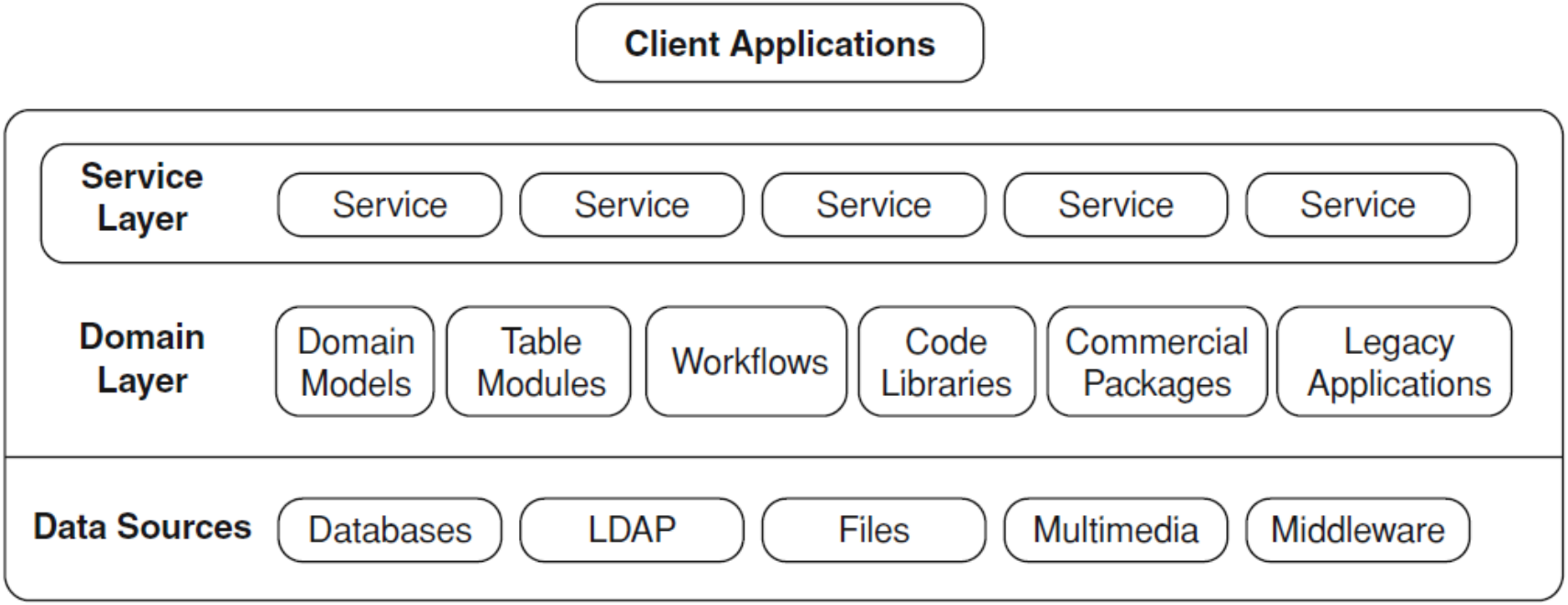
REMOTE PROCEDURE CALL

- Objects were frequently used in distributed scenarios
- When a client invoked a method on the proxy's interface, the proxy would dispatch the call over the network to a remote stub, and the corresponding method on the distributed object would be invoked
- As long as the client and distributed object used the same technologies, everything worked pretty well



WEB SERVICES

- Web services help to insulate clients from the logic used to fulfill their requests.
- They establish a natural layer of indirection that makes it possible for clients and domain entities to evolve independently



RESOURCE BASED ARCHITECTURES

- Resources are Representations of Real World Entities
  - E.g. People, Invoices, Payments, etc.
  - Relationships are typically nested
  - Hierarchies or Webs…not Relational Models
- Resources are Represented in URIs
  - URIs are Paths to Resources
    - Query Strings for non-data elements, e.g. format, sorting, etc.

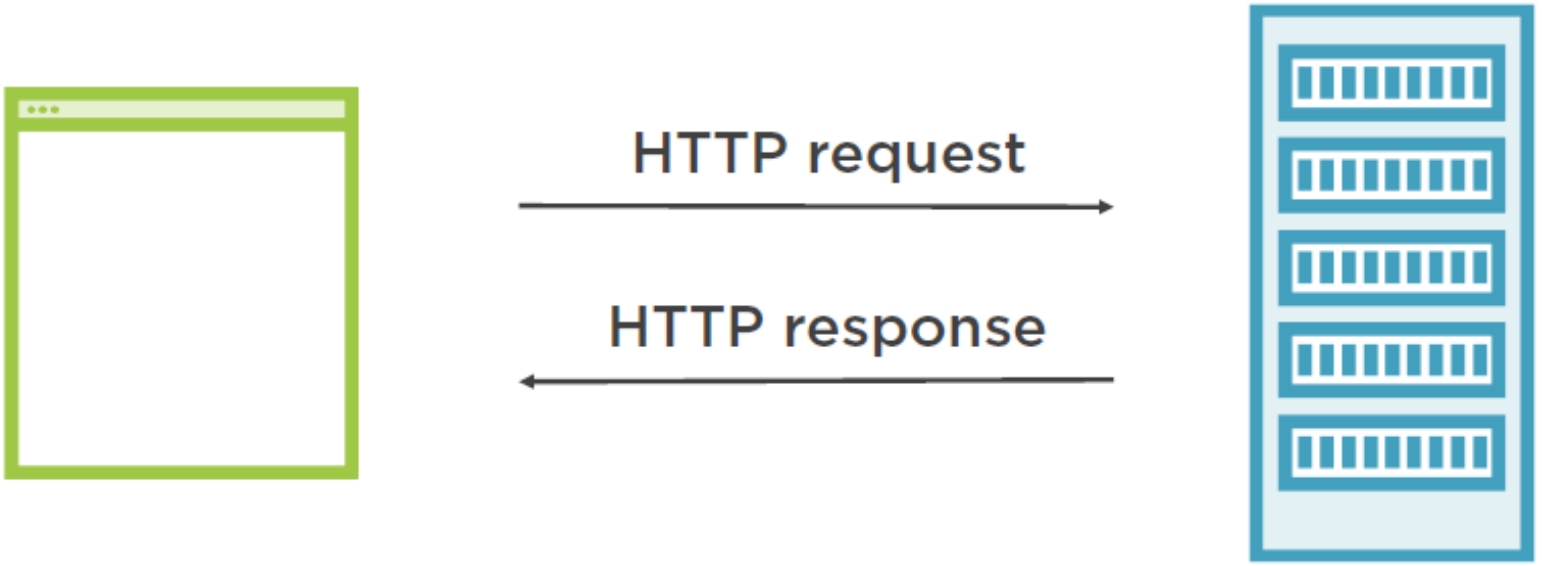
REPRESENTATIONAL STATE TRANSFER (REST)

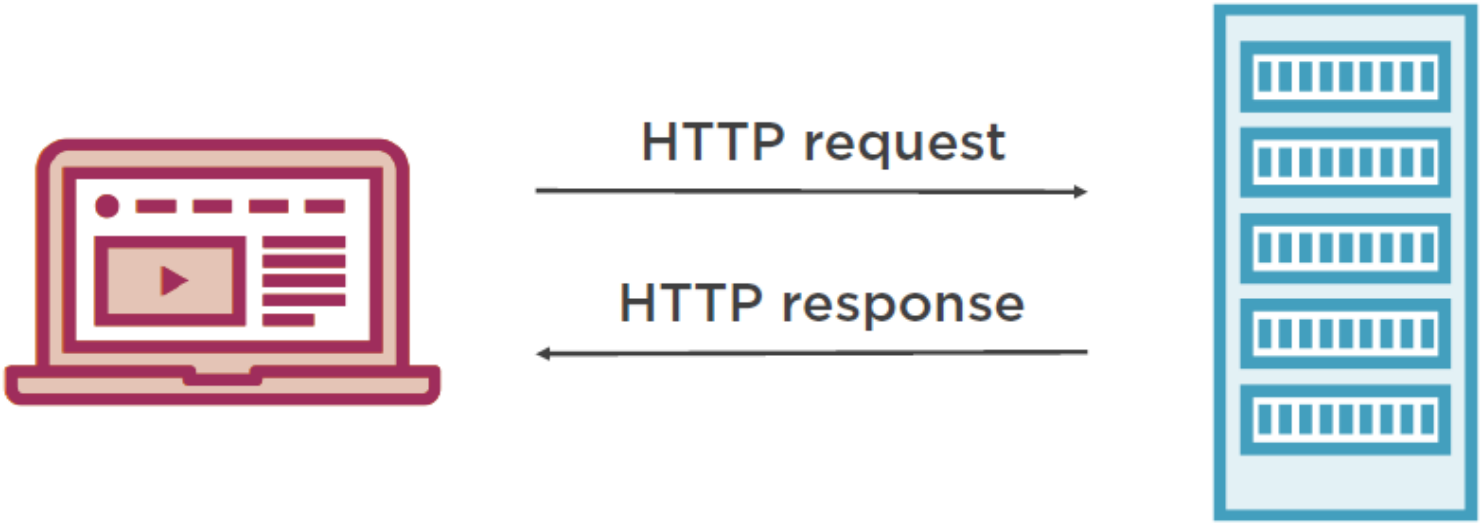
- Representational State Transfer is intended to evoke an image of how a well-designed web application behaves:
  - a network of web pages (a virtual state-machine)…
  - … where the user progresses through an application by selecting links (state transitions)…
  - … resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use
- REST is an architectural style – not a standard
- REST is protocol agnostic

Roy Fielding

http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

REPRESENTATIONAL STATE TRANSFER





REPRESENTATIONAL STATE TRANSFER

SIX CONSTRAINTS OF REST

Client-Server

client and server are separated

(client and server can evolve separately)

Statelessness

state is contained within the request

Cacheable

each response message must explicitly state if it can be cached or not

Layered System

client cannot tell what layer it's connected to

Code on Demand (optional)

server can extend client functionality

Uniform Interface

API and consumers share one single, technical interface: URI, Method, Media Type

UNIFORM INTERFACE SUBCONSTRAINTS

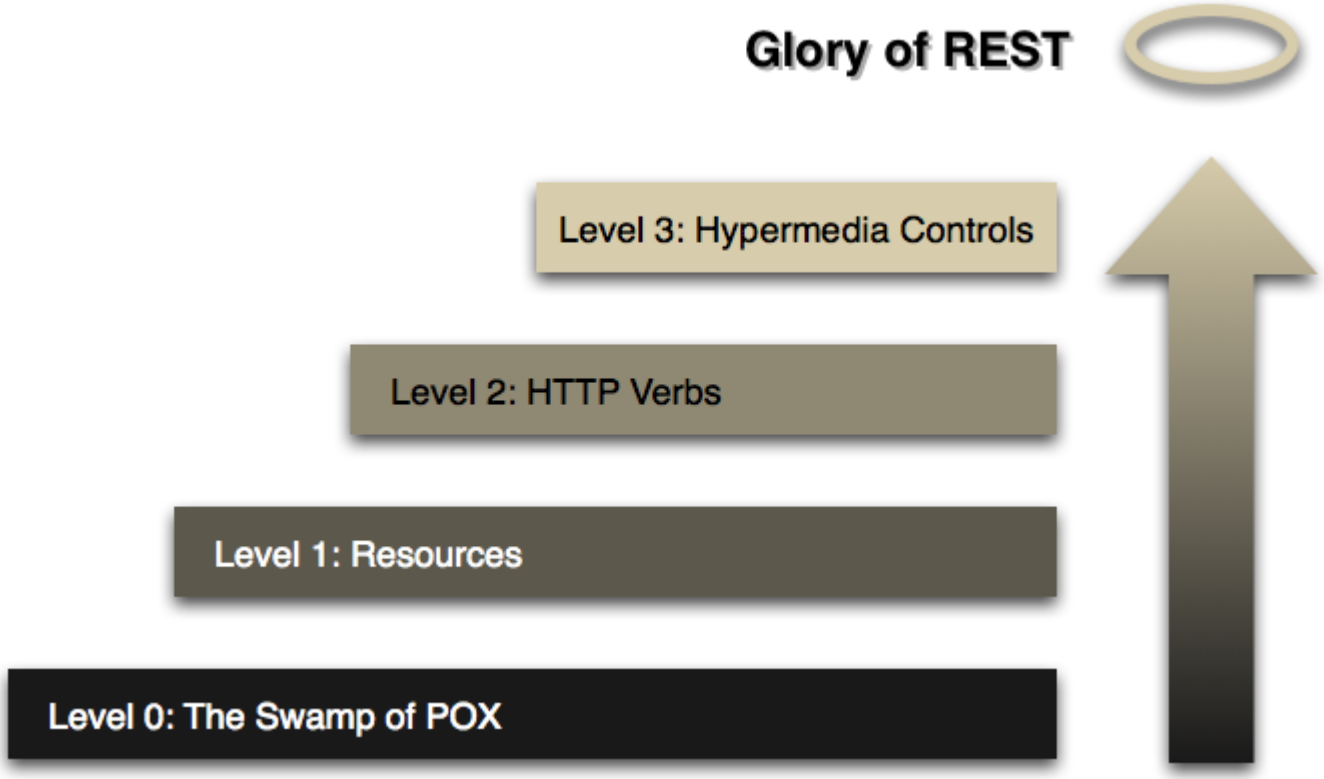
- Identification of resources
  - A resource is conceptually separate from its representation
  - Representation media types: application/json, application/xml, custom, …
- Manipulation of resources through representations
  - Representation + metadata should be sufficient to modify or delete the resource
- Self-descriptive message
  - Each message must include enough info to describe how to process the message
- Hypermedia as the Engine of Application State (HATEOAS)
  - Hypermedia is a generalization of Hypertext (links)
  - Drives how to consume and use the API
  - Allows for a self-documenting API

THE RICHARDSON MATURITY MODEL

- Level 0: Swamp of POX
  - Tunnels requests and responses through its protocol without using the protocol to indicate application state.
- Level 1: Resources
  - Distinguish between different resources, it might be level 1.
- Level 2: HTTP verbs
  - This level indicates that your API should use the protocol properties in order to deal with scalability and failures
- Level 3: Hypermedia controls
  - Uses HATEOAS to deal with discovering the possibilities of your API towards the clients

http://restcookbook.com/Miscellaneous/richardsonmaturitymodel/

THE RICHARDSON MATURITY MODEL



https://martinfowler.com/articles/richardsonMaturityModel.html

ASP.MVC CORE

ASP.NET CORE AND THE MODERN WEB


Totally Modular


Faster Development Cycle


Seamless transition
from on-premises to cloud


Choose your Editors
and Tools

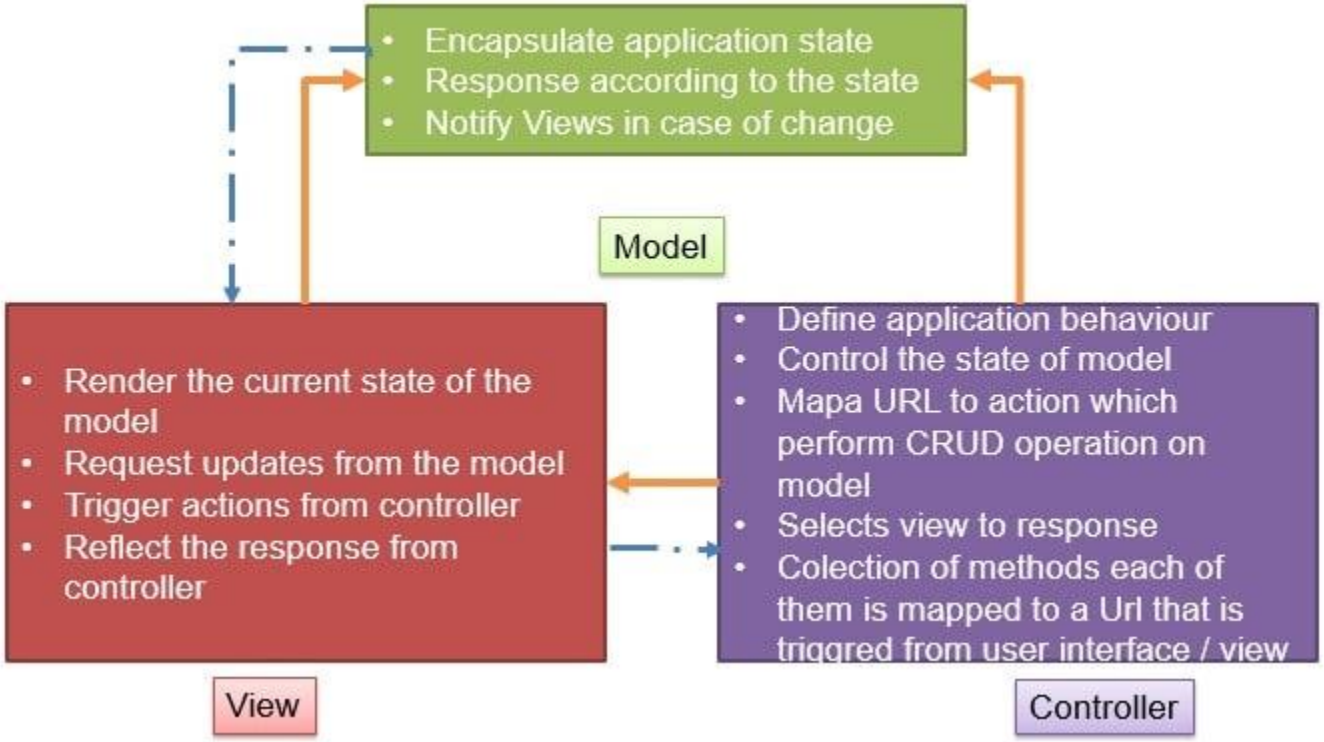
Open Source
with Contributions


Cross-Platform


Fast

MVC ARCHITECTURE

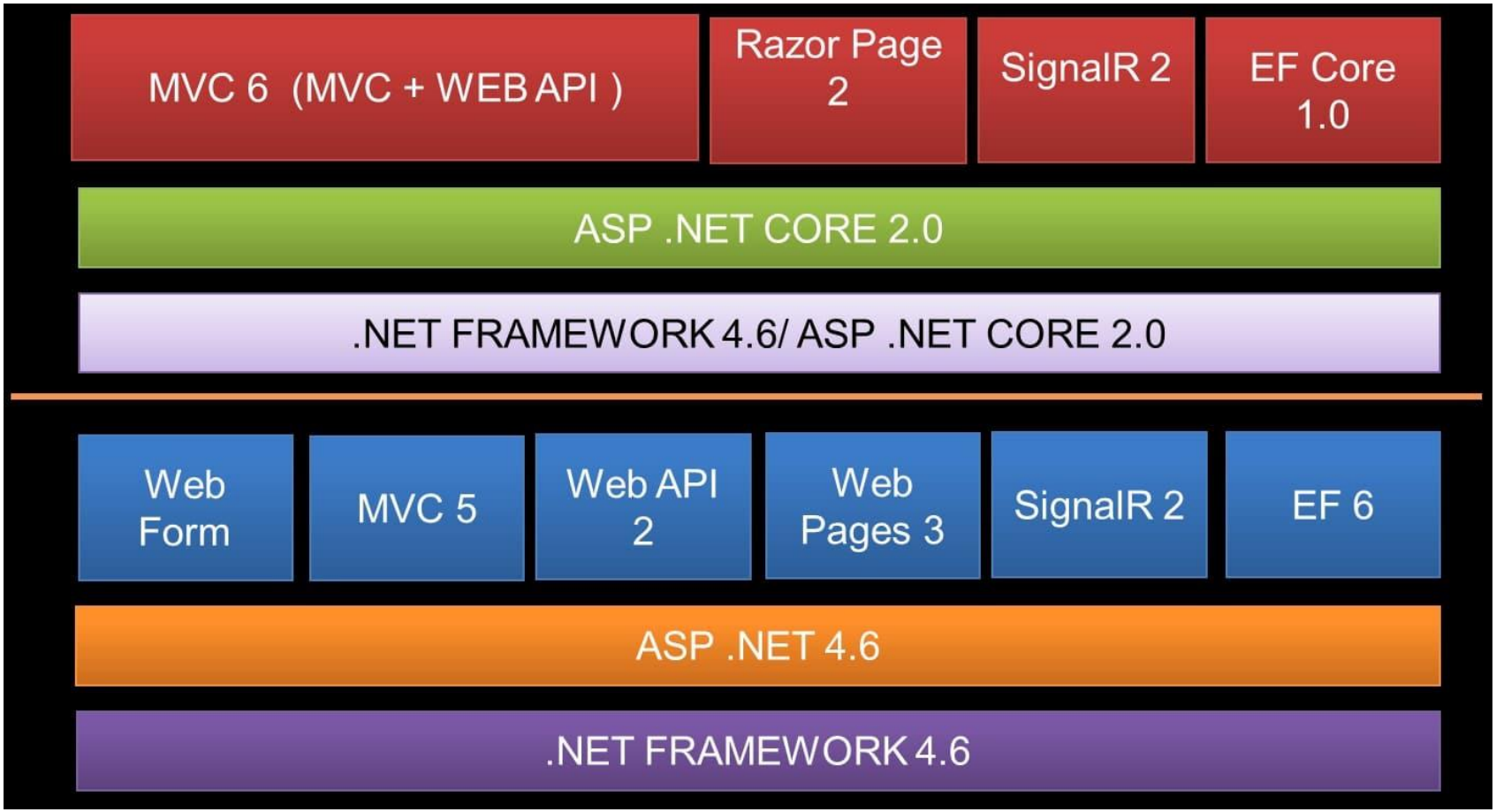
- The Model-View-Controller (MVC) pattern is an architectural design principle
- Separates the application components of a Web application into three layers.



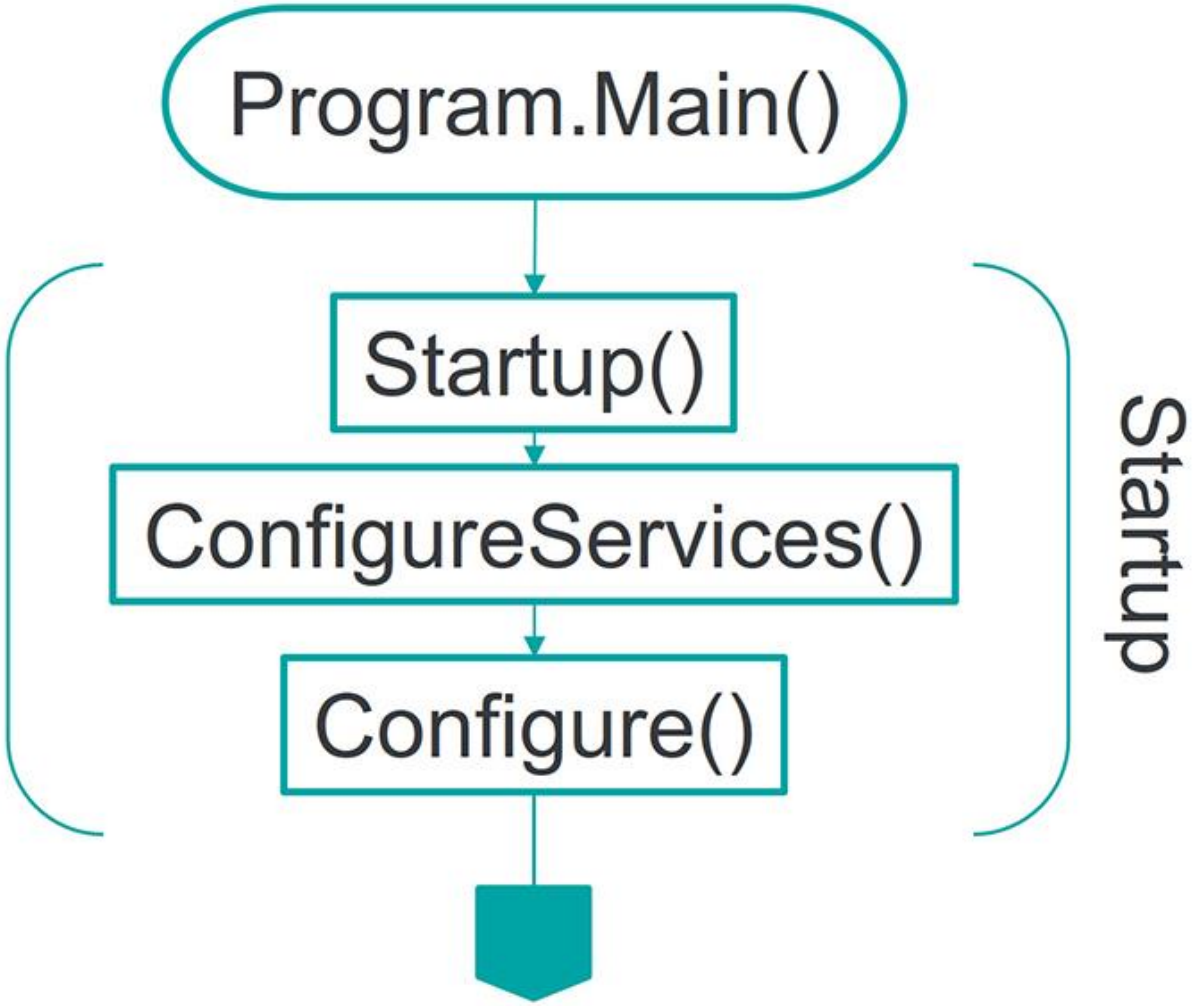
MVC ARCHITECTURE

- **The Model** is the part of the application that handles the logic for the application data. Often model objects retrieve data (and store data) from a database
- **The View** is the parts of the application that handles the display of the data.
  Most often the views are created from the model data
- **The Controller** is the part of the application that handles user interaction

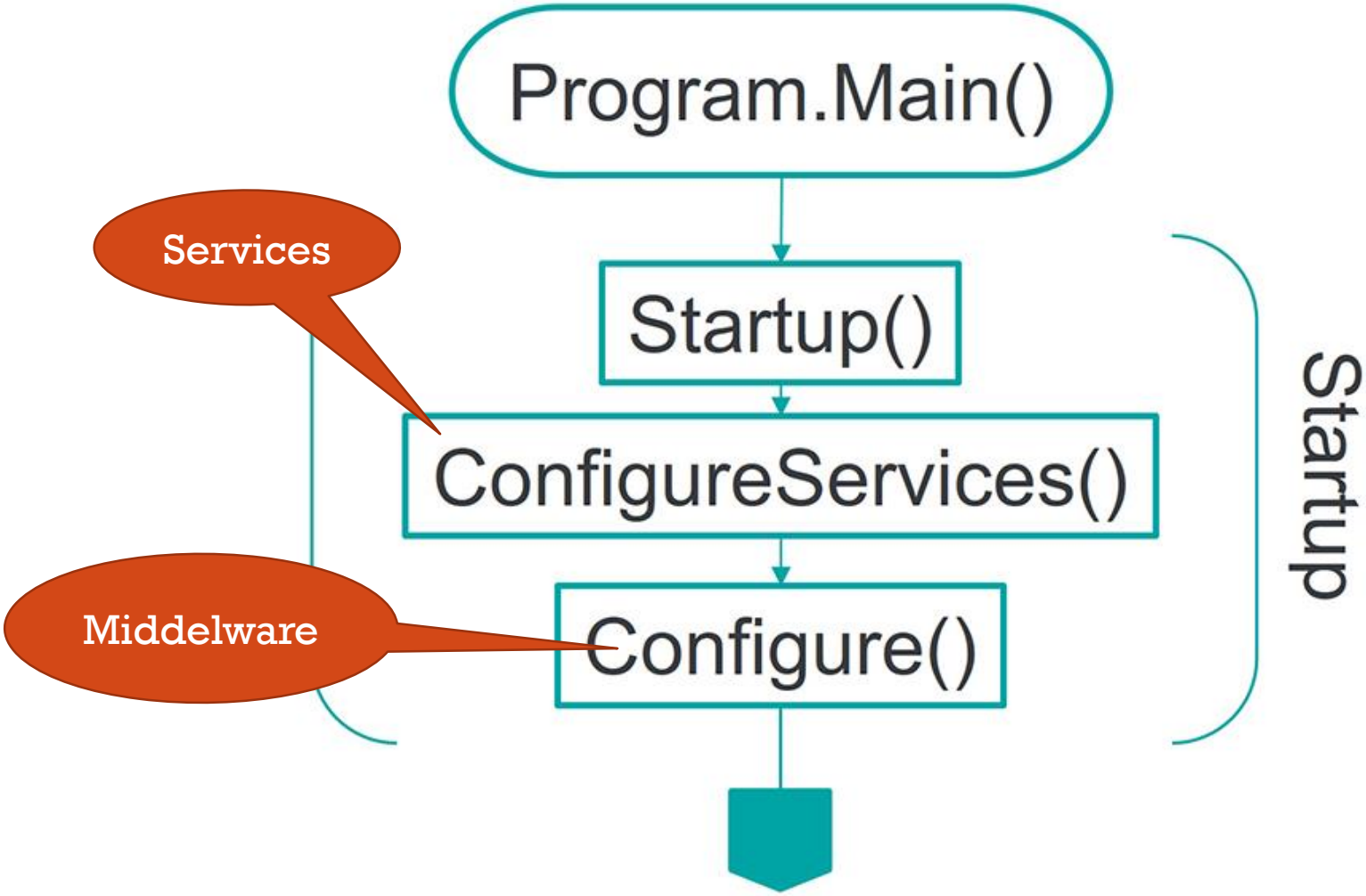
ASP.NET CORE 2.0 MVC 6

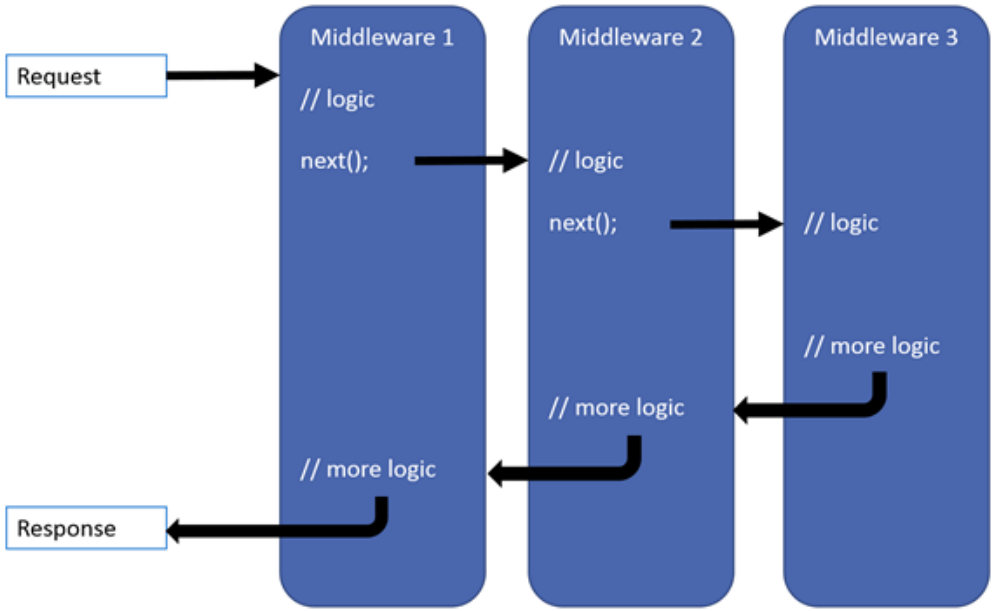


MIDDLEWARE & SERVICES



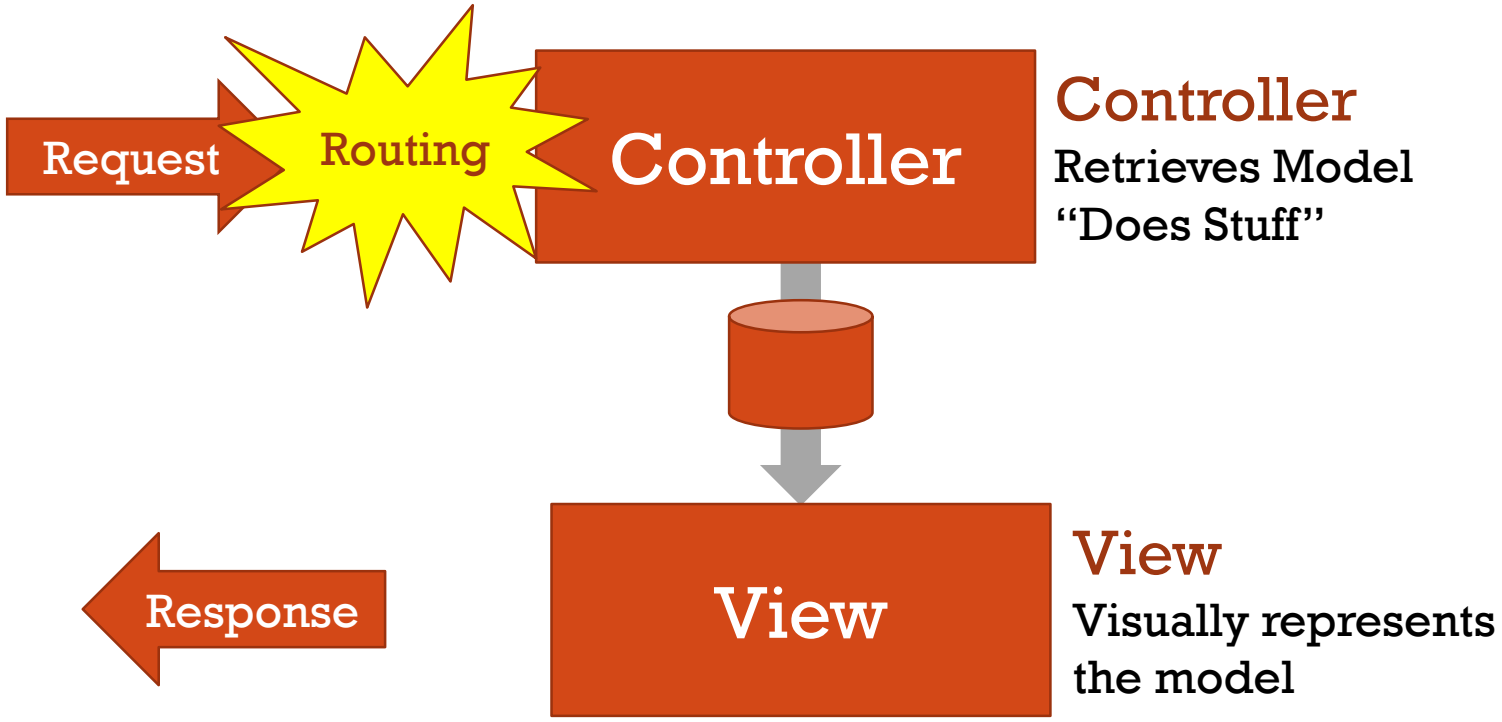
MIDDLEWARE & SERVICES





WHAT IS MIDDLEWARE?

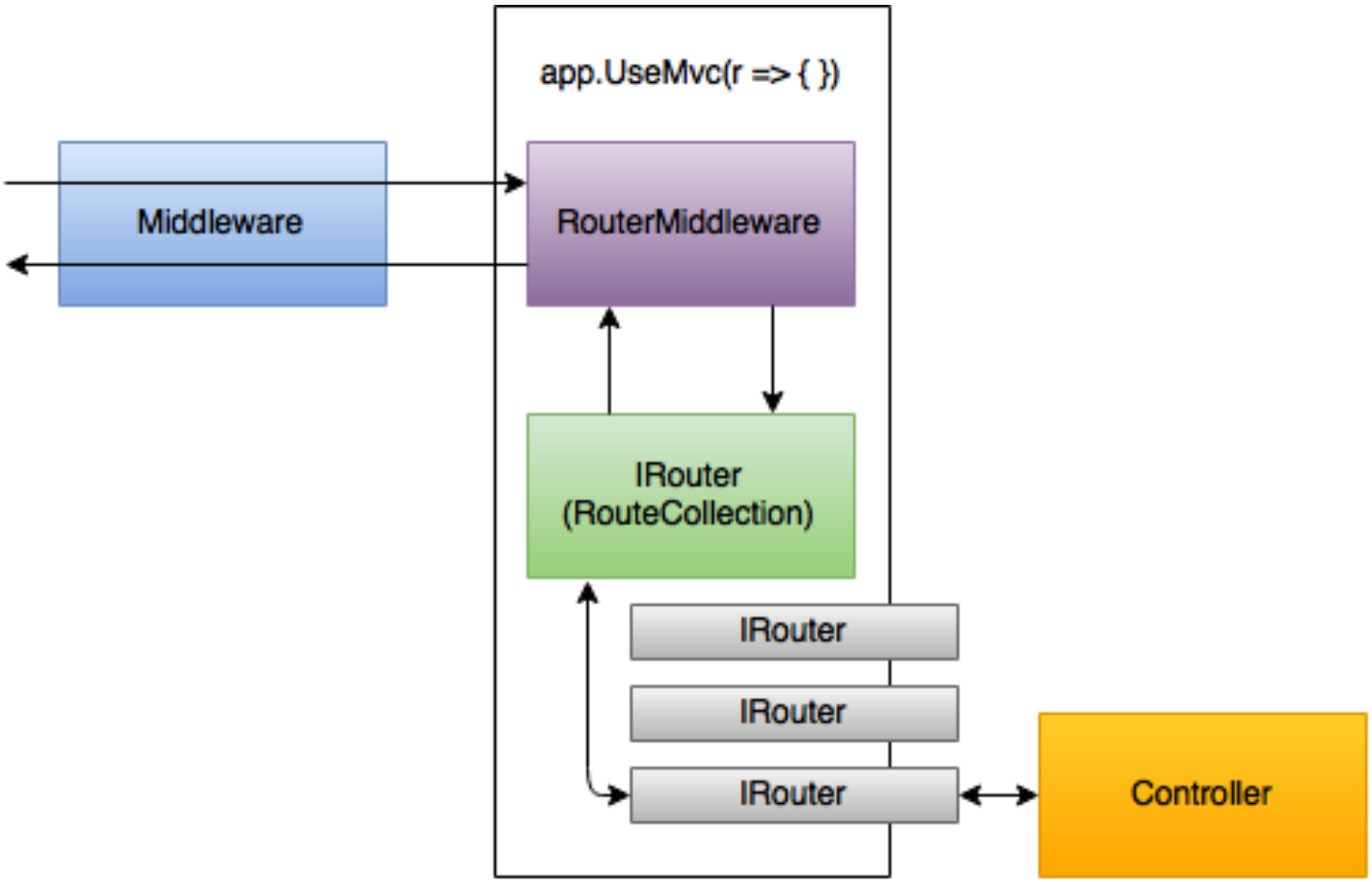
How MVC Works



Controller
Retrieves Model
"Does Stuff"

View
Visually represents
the model

ROUTING


app.UseMvc(r => { })
Middleware
RouterMiddleware
IRouter
(RouteCollection)
IRouter
IRouter
IRouter
Controller

WORKING WITH ROUTING

- Convention-based Routing

```
app.UseMvc(routes =>{
     routes.MapRoute(
          name: "default",
          template:
     "{controller=Values}/{action=Index}/{id?}");
});
```

- Attribute-based routing
  - Use attributes at controller and action level: [Route], [HttpGet], …

- The ASP.NET Core team advises to use attribute-based routing for APIs

UNIFORM INTERFACE API

TOOLS

https://www.getpostman.com/

https://www.google.com/chrome

+ JSON Formatter extension

STRUCTURING THE OUTER FACING CONTRACT

Resource Identifier

HTTP Method

Payload

(representation: media types)

RESOURCE NAMING

- Nouns: things, not actions
  - ~~api/getCategory~~
  - GET api/categories
  - GET api/categories/{categoryId}
- Represent hierarchy when naming resources
  - api/products/{productid}/categories
  - api/products/{productid}/categories /{categoryId}
- Filters, sorting orders, … aren't resources
  - ~~api/categories/sortBy/name~~
  - api/categories?sortBy=name

RESOURCE NAMING

- Resources are often named using an auto numbered DB field as part of their URI
  - Resource URIs should remain the same
- GUIDs can be used instead
  - Allows switching out backend data stores
  - Potentially hides implementation details

OUTER FACING CONTRACT

- The Outer Facing Contract is NOT your entity/domain model
  - Even if they are identical in structure, they are semantically VERY different
  - Decoupling between layers in the system
  - Outer Facing Contract is what users of your API will know and use

METHOD SAFETY AND METHOD IDEMPOTENCY

- A method is considered safe when it doesn't change the resource representation
- A method is considered idempotent when it can be called multiple times with the same result

HTTP	Method Safe?	Idempotent?
GET	yes	yes
HEAD	yes	yes
POST	no	no
DELETE	no	yes
PUT	no	yes
PATCH	no	no

THE IMPORTANCE OF STATUS CODES

Level 200 - Success
200 – Ok
201 – Created
204 – No content

Level 400 – Client Mistakes
400 – Bad request
401 – Unauthorized
403 – Forbidden
404 – Not found
405 – Method not allowed
406 – Not acceptable
409 - Conflict
415 – Unsupported media type
422 – Unprocessable entity

Level 500 Server Mistakes
500 – Internal server error

ERRORS VERSUS FAULTS

Errors

- Consumer passes invalid data to the API, and the API correctly rejects this
- Level 400 status codes

Faults

- API fails to return a response to a valid request
- Level 500 status codes

HTTP METHOD OVERVIEW BY USE CASE

Reading resources

- GET api/categories
  - 200 Ok [{category},{category}]
  - 404 Not found

- GET api/ categories /{Id}
  - 200 Ok {category}
  - 404 Not found

Deleting resources

- DELETE api/ categories /{Id}
  - 204 No content
  - 404 Not found

HTTP METHOD OVERVIEW BY USE CASE

Creating resources

- POST api/categories – {category}
  - 201 Created {category}
  - 404 Not found
- *POST api/*categories */{Id} can never be successful*
  - 404 Not found
  - 409 - Conflict

Updating resources

- PUT api/categories/{Id} – {category}
  - 200 Ok {category}
  - 204 No content
  - 404 Not found
- PATCH api/categories/{Id} - {JsonPatchDocument on category}
  - 200 Ok {category}
  - 204 No content
  - 404 Not found

UPDATING A RESOURCE

- HTTP PUT updates full resource
- HTTP PATCH is for partial updates
- The request body of a patch request is described by RFC 6902 (JSON Patch)
  - https://tools.ietf.org/html/rfc6902
- Patch requests should be sent with media type application/json-patch+json
  - But most APIs accept also application/json

JSON PATCH OPERATIONS

Add

```
{"op": "add",
"path": "/a/b",
"value": "foo"}
```

Remove

```
{"op": "remove",
"path": "/a/b"}
```

Replace

```
{"op": "replace",
"path": "/a/b",
"value": "foo"}
```

Copy

```
{"op": "copy",
"from": "/a/b",
"path": "/a/c"}
```

Move

```
{"op": "move",
"from": "a/b",
"path": "/a/c"}
```

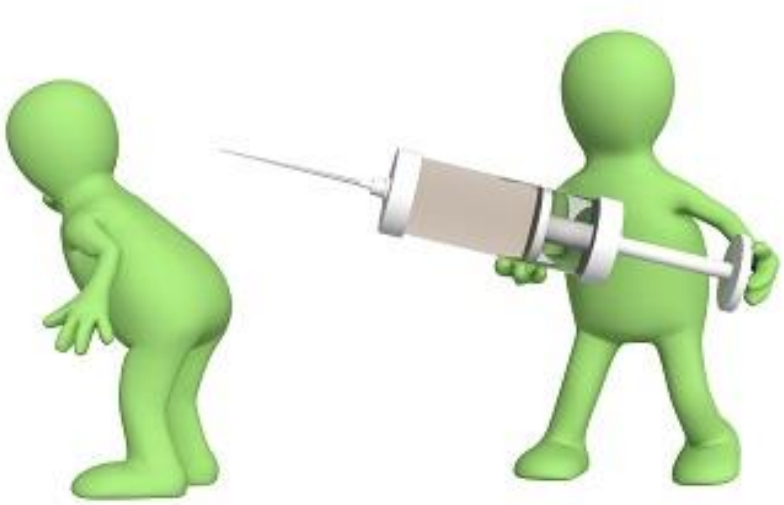
Test

```
{"op": "test",
"path": "/a/b",
"value": "foo"}
```

PATCH EXAMPLE

```
[
    {
        "op": "replace",
        "path": "/title",
        "value": "new title"
    }
    ,
    {
        "op": "remove",
        "path": "/description"
    }
]
```

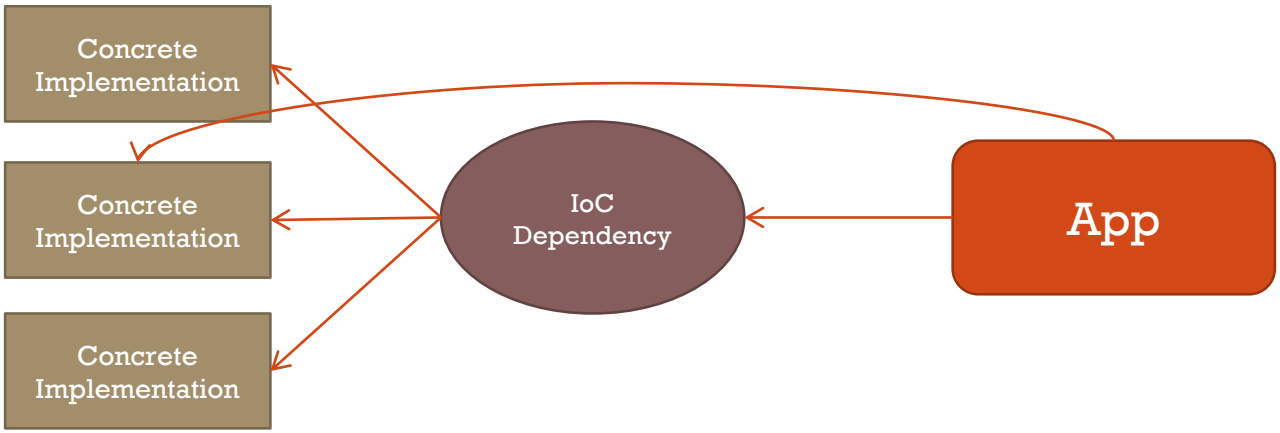
- array of operations
- "replace" operation
- "title" property gets value "new title"
- "remove" operation
- "description" property is removed (set to its default value)



DEPENDENCY INJECTION

WHAT IS DEPENDENCY INJECTION

- A software design pattern that implements inversion of control for resolving **dependencies**.
  A **dependency** is an object that can be used (a service). An **injection** is the passing of a **dependency** to a dependent object (a client) that would use it.
  - Wikipedia



TESTABILITY