CMPUT 401 Software Process and Product Management

Ildar Akhmetov

ildar@ualberta.ca

Department of Computing Science

University of Alberta

Software Architecture

Fall 2020

So, we must design a new system: Where do we start?

(1) From scratch

= Top-down

= Greenfield engineering

(2) From a bunch of pre-existing software

= Bottom-up

= Brownfield engineering

Brownfield

Pros:

- Existing code can be reused
- There is a starting point
- Incremental improvements
- Already defined and documented processes

Cons:

- Legacy code can bring problems
- Detailed understanding of existing systems is required
- Some pieces of existing systems may need redesign

Greenfield

Pros:

- Possible to use the best technologies available
- No legacy constraints or dependencies

Cons:

- Risk is higher due to the lack of clear direction
- More time needed
- Difficult to make critical decisions

Greenfield vs Brownfield

Aspect	Greenfield	Brownfield
Project direction	Vague	Clear
Development effort	Comparatively more since everything needs to be build from scratch	Comparatively less since basic foundation is already built
Dependency on older systems	No	Substantial
Development time	Comparatively more	Comparatively less
Degree of risk	Comparatively higher	Comparatively lower
Re-engineering required	No	Likely
Costs	Can be costly if there is no clear direction	Can be costly due to the presence of legacy code

Source: https://synoptek.com/insights/it-blogs/greenfield-vs-brownfield-software-development/

Tiers and Layers

Layers

- conceptual elements, organizing the types of functionalities that must exist in any information system

Tiers

- correspond to computational elements

Layers

*Example: **MVC**:*
Model–view– controller

Presentation

acquiring (delivering) information from (to) the user or external systems

Application logic

information processing to deliver functionalities (services)

Data/Resources

management of persistent content

Tiers

*How are **layers** distributed within system?*

1 tier

- Monolithic architecture

2 tiers

- Client-Server architectures

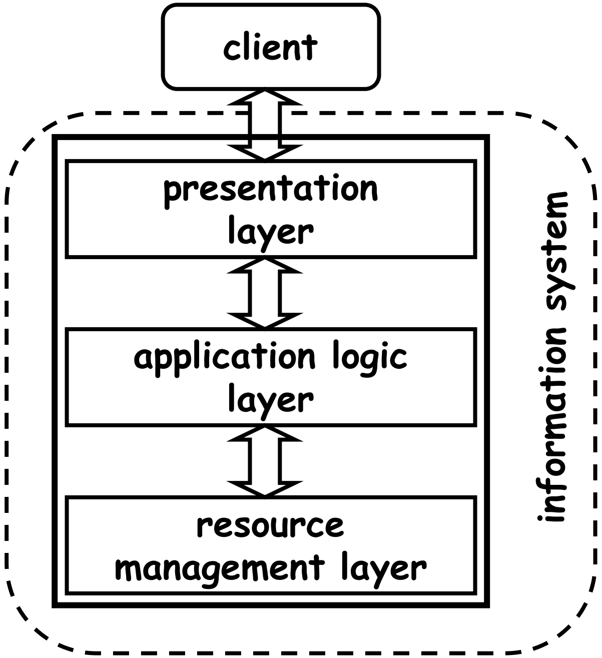
3 tiers

- 3-tier architectures

1-Tier Architecture

The obvious original architecture of mainframes and dumb terminals

- Simple deployment: no need to develop or maintain clients
- No APIs:
  - No need for complex data transformations
  - Reuse through screen scraping
- High performance
  - No need for portability, native-system code
  - All layers execute within the same context: no indirection
- Difficult to maintain



2-Tier (Client-Server) Systems

Server (bottom tier) fulfills requests by clients. Server tasks:

- query execution,
- data-integrity management,
- business logic,
- resource management.

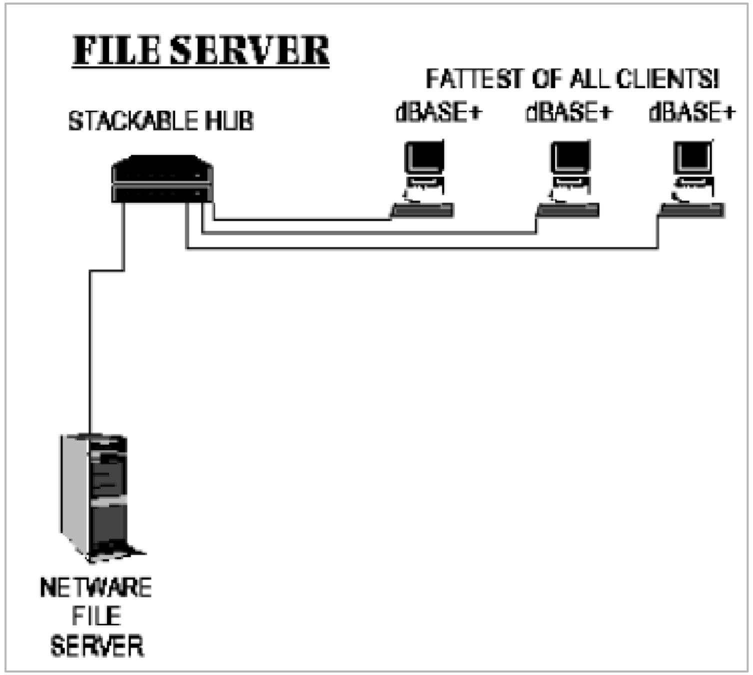
Clients (top tier) make requests to servers. Client tasks:

- data entry and validation,
- query issuing,
- workflow enactment.

File Server

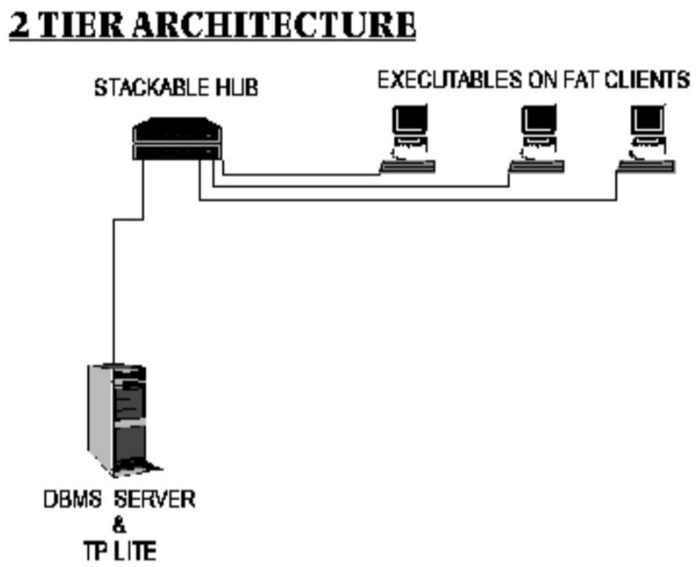
Examples: dBase, FoxPro, Clipper, Clarion

- Many terminals access files off a common file system
- "Fat" clients (logic and presentation reside on the client)
- Server and clients exchange files
- Assumes low usage, infrequent file transfer



Database Server

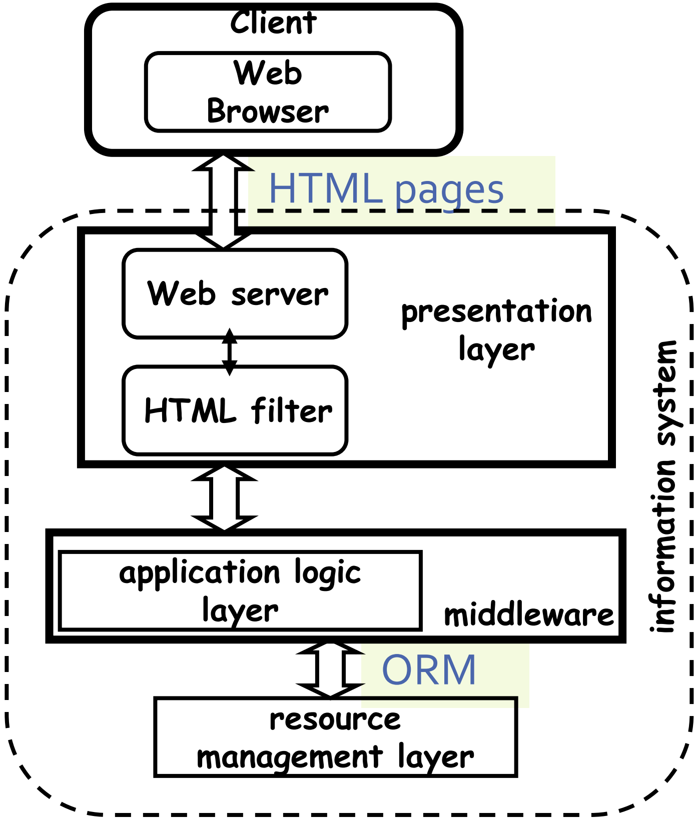
- Many PCs (with GUIs) send queries to the central database, using RPC or SQL
- Some logic moves to the server: the DB server may also provide
  - Procedures
  - Triggers
  - Query planning



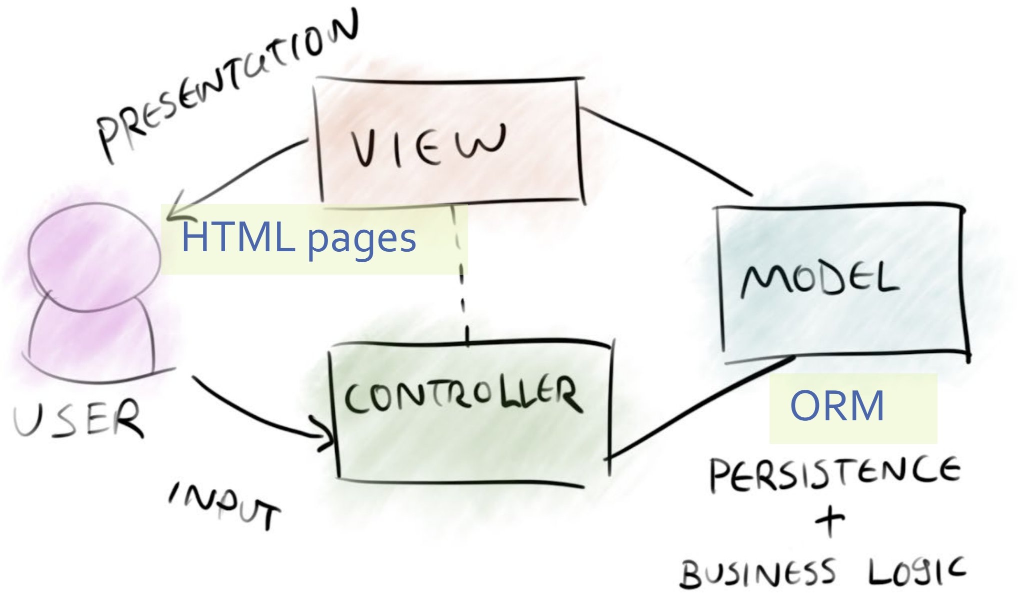
Old 3-tier Architecture For Web Apps

Presentation layer is viewed as two components:

- Web server, to communicate with the client browser
- HTML filter to construct the HTML pages to transmit to the browser



MVC



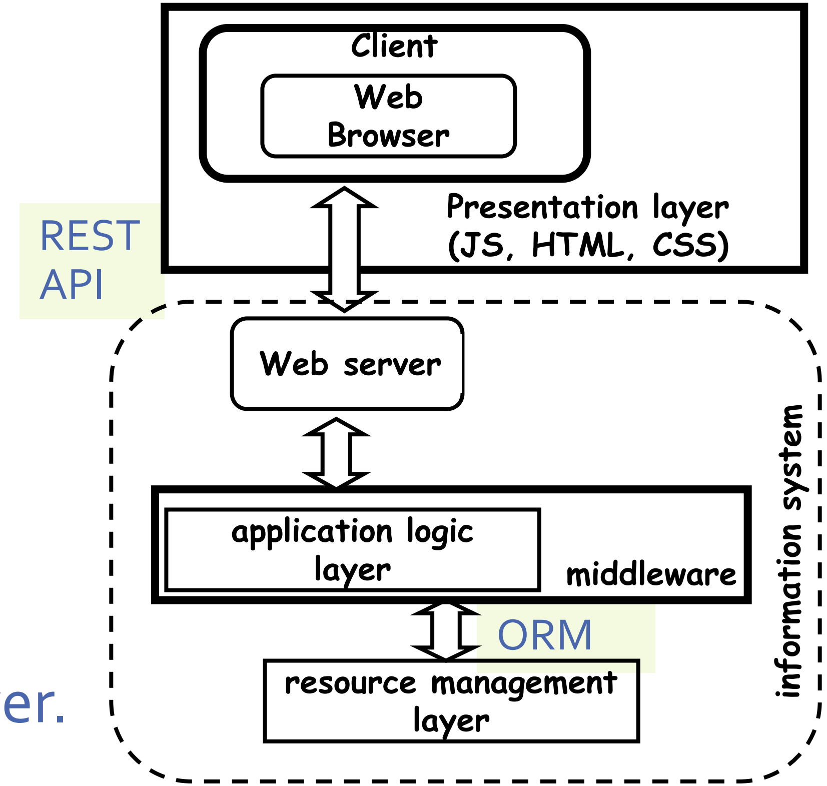
Source: https://synoptek.com/insights/it-blogs/greenfield-vs-brownfield-software-development/

REST-based SOA 3-tier Architecture

Presentation layer is at the browser

- HTML for the overall layout
- CSS for colors and fonts,
- JavaScript for generating dynamic content

Middleware constructs JSON/XML data representations, sent to the browser through the web server.



References:

"Web Services: Concepts, Architecture and Applications" By G. Alonso, F. Casati, H. Kuno, V. Machiraju
http://www.amazon.com/Web-Services-Gustavo-Alonso/dp/3540440089

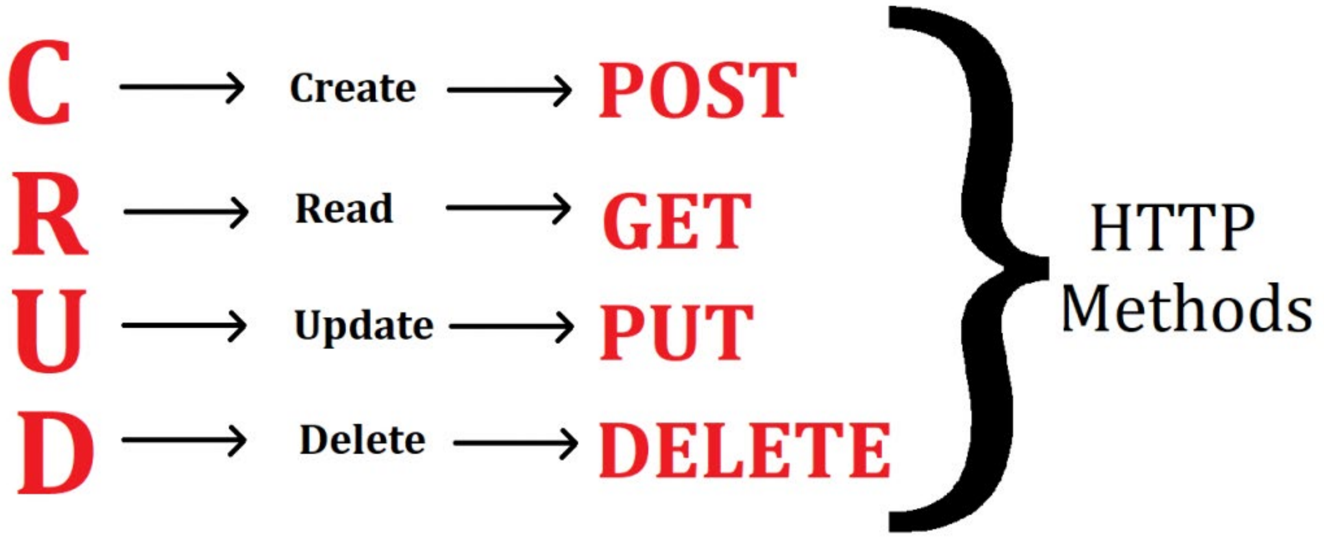
Client/Server Past, Present, and Future, by Schussel, George (1995).http://ciains.info/elearning/Solutions/Architecture/ClientServer/CS-past,presentFurure.pdf

REST Rules

The de facto standard!

1. Use HTTP methods explicitly
2. Return correct status codes
3. Be stateless
4. Have clear URIs
5. Support multiple data-exchange representations

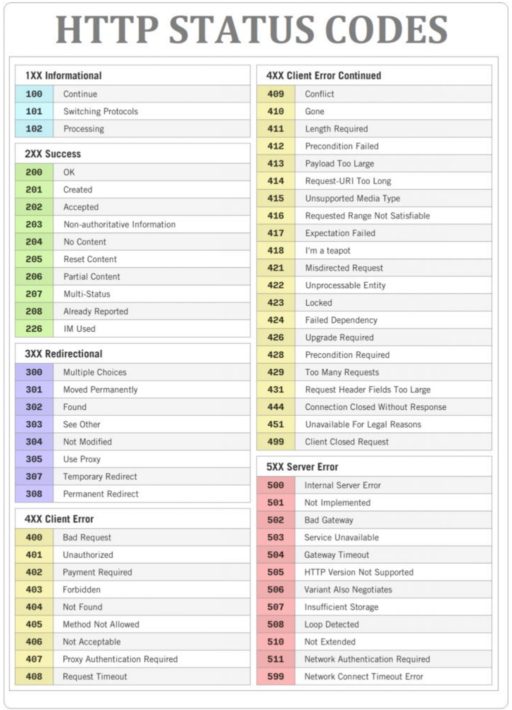
1) Use HTTP methods explicitly



Source: https://bytenbit.com/best-guidelines-design-restful-api/

2) Return correct status codes

- 2xx – Success
- 3xx – Redirection
- 4xx – Client error
- 5xx – Server error



HTTP STATUS CODES

1XX Informational	
100	Continue
101	Switching Protocols
102	Processing

2XX Success	
200	OK
201	Created
202	Accepted
203	Non-authoritative Information
204	No Content
205	Reset Content
206	Partial Content
207	Multi-Status
208	Already Reported
226	IM Used

3XX Redirectional	
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
308	Permanent Redirect

4XX Client Error	
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout

4XX Client Error Continued	
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Payload Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
418	I'm a teapot
421	Misdirected Request
422	Unprocessable Entity
423	Locked
424	Failed Dependency
426	Upgrade Required
428	Precondition Required
429	Too Many Requests
431	Request Header Fields Too Large
444	Connection Closed Without Response
451	Unavailable For Legal Reasons
499	Client Closed Request

5XX Server Error	
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
506	Variant Also Negotiates
507	Insufficient Storage
508	Loop Detected
510	Not Extended
511	Network Authentication Required
599	Network Connect Timeout Error

Source: https://bytenbit.com/best-guidelines-design-restful-api/

Some HTTP Status Codes for REST API

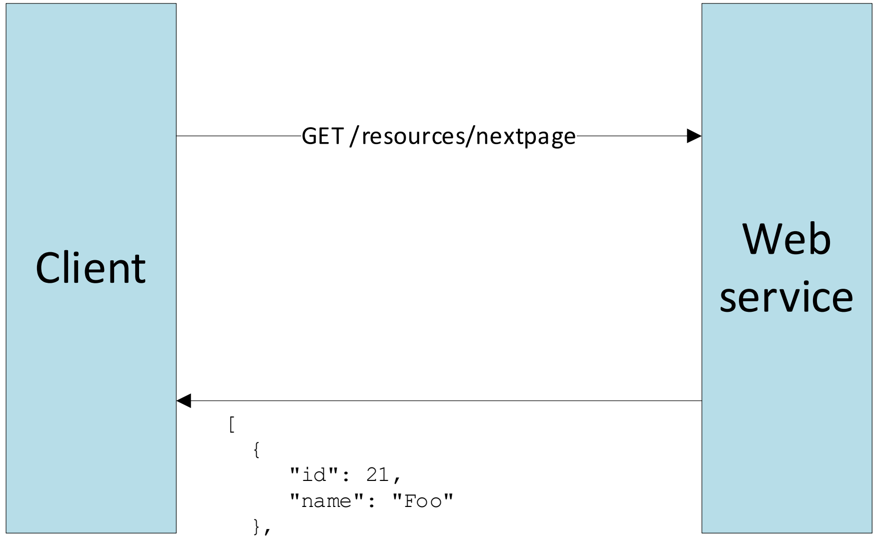
- 200 – OK
- 201 – Created
- 400 – Bad Request
- 401 - Unauthorized
- 404 – Not Found
- 405 – Method Not Allowed
- 500 – Internal Server Error

3) Be Stateless

- Every HTTP request happens in complete isolation
- When the client makes an HTTP request, it includes all information necessary for the server to fulfill that request
- The server never relies on information from previous requests
- If that information was important, the client would have sent it again in this request

Source: https://restfulapi.net/statelessness/

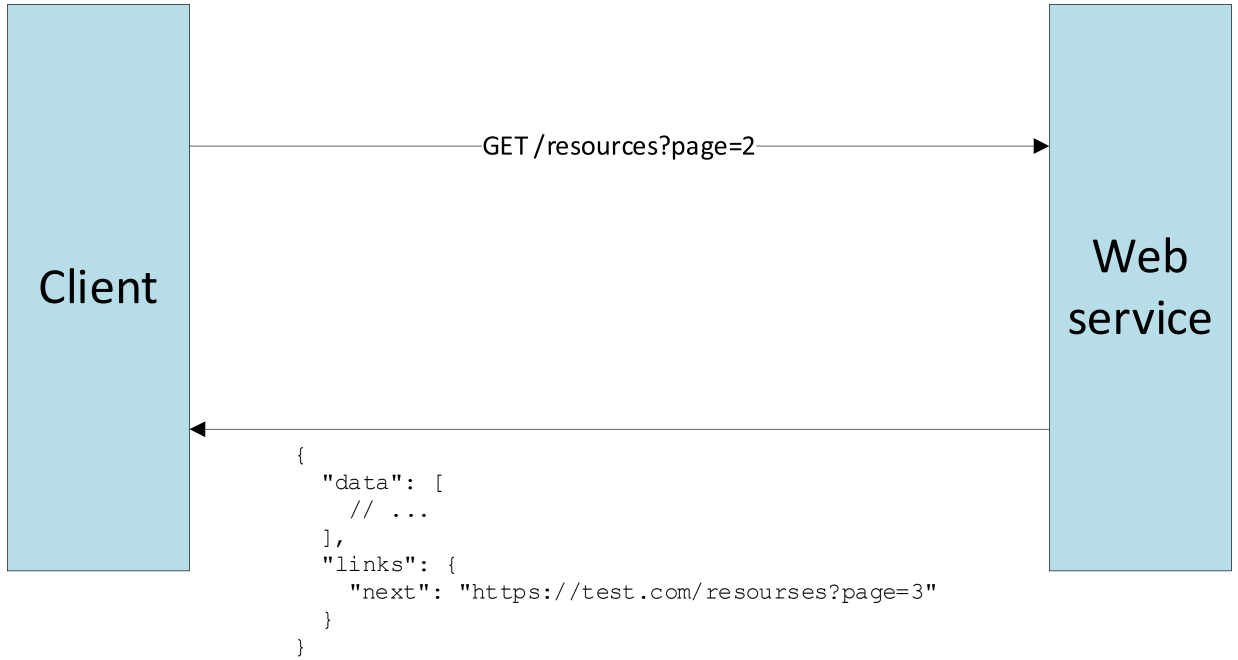
Stateful API



```
next_page = previous_page + 1
return next_page
```

```
[
  {
     "id": 21,
     "name": "Foo"
  },
  {
     "id": 22,
     "name": "Bar"
  }
]
```

Stateless API



```
{
  "data": [
    // ...
  ],
  "links": {
    "next": "https://test.com/resourses?page=3"
  }
}
```

4) Have Clear URIs

- Use two URIs per resource
- Use plural nouns
- Don’t use verbs
- Use lower case
- Use hyphens, not underscores
- Avoid file extensions

```
/books                # All books
/books/57             # Single book
```

```
GET /books                 # All books
GET /books?status=sold     # Param
POST /books                # Create
PUT /books/82              # Update
```

```
/getAllBooks
/getAllSoldBooks
/createBook
/updateBook
```

Quiz

- Available on eClass
- Submit until the end of this week (Sunday 11:59 pm)