Web Application
is a client-server computer program in which the client (including the user interface and client-side logic) runs in a web browser.
GMAIL or facebook message, google map etc

Web Service
is a software system designed to support interoperable machine-to-machine interaction over a network.
In a Web service, Web technology such as HTTP, originally designed for human-to-machine communication, more specifically for transferring machine-readable file formats such as XML and JSON.

Webサービスとは、大まかにいえば、Webの通信の仕組みを利用して、コンピュータ同士でさまざまなデータをやり取りし、分散処理を行うシステムです。

例えば、ショッピング・サイトのAmazon.comでは、「Product Advertising API」というWebサービスを提供しています。このWebサービスを使うと、Amazon.comで扱っている商品を検索したり、ショッピング・カートを操作したりといった機能を、オリジナルのアプリケーションに組み込むことができます。

https://widgetsrus.com/api/v2/orders?search=flobulator+mk+4&count=25
Protocol
Hostname
basepath
Query parameters

Concurrency (並列性) vs Parallelism (平行性)
- Concurrency is about dealing with lots of things at once.
- Parallelism is about doing lots of things at once.

Cascading Style Sheets (CSS)
is a language that describes the style of an HTML document. CSS describes how HTML elements should be displayed.

In CSS, a full stop indicates a class and a hash symbol indicates an ID.

```
. first_section{
   background - color :  red;
   text  -  align :  center;
}

#second_section{
   font - weight :
   bold;
}
```

The reason of why JavaScript file should be included just before the closing tags.

- no need to have a check if the DOM is loaded, since by having the scripts at the end.
- If the JavaScript files are included at the top of the document, it will be a visual delay before the end user sees the actual page. This is completely avoided if you include the JavaScript files at the end of the document.

Put Stylesheets at the top

Including stylesheet in the head elements makes pages appear to be loading faster. it allows the page to render progressively.
The problem with putting stylesheets near the bottom of the document is that it prohibits progressive rendering in many browsers.

CRUD

- To **C**reate a resource on the server, use POST
- To **R**etrieve a resource, use GET
- To change the state of a resource or to **U**pdate it, use PUT
- To remove or **D**elete a resource, use DELETE

REpresentational State Transfer (REST)

is Platform-independent

- Server is Unix, client is a Mac

Language-independent

- C# can talk to Jave, ext.

Standards-based (runs on top of HTTP*),
Can easily be used in the presence of firewalls.

Resources

Nouns not verbs
therefore
GET /adduser?name=Robert HTTP/1.1
is pretty messy
because using GET for adding user and should not user verb for resources
so it should be something like

```
POST /users HTTP/1.1
Host: myserver
Content-Type: application/json
{
        “name”: “Robert”
}
```

RESTの原則

- Addressability
  - 提供する情報がURIを通して表現できること。全ての情報はURIで表現される一意なアドレスを持っていること。
- Stateless
  - HTTPをベースにしたステートレスなクライアント/サーバプロトコルであること。セッション等の状態管理はせず、やり取りされる情報はそれ自体で完結して解釈できること。
- Connectability
  - 情報の内部に、別の情報や(その情報の別の)状態へのリンクを含めることができること。
- Uniform Interface
  - 情報の操作(取得、作成、更新、削除)は全てHTTPメソッド(GET、POST、PUT、DELETE)を利用すること。

RESTful

typically but not always, communicate over the HTTP with the same HTTP verbs (GET, POST, PUT, DELETE, etc.) used by web browser to retrieve web pages and send data to remote servers.

メリット

1. URIに規律が生まれることで、APIを利用するサービス開発者が楽になる。
2. URIに規律が生まれることで、API開発者もURIからソースのどの部分なのかが容易にわかる。
3. ブラウザのアドレスバーにURIを入力すればリソースが参照できる。
4. server, client 間で何も共有しないことにより、付加に応じたscalability が向上する。
5. GET, POST, PUT, DELETE等のHTTP標準のメソッドを使うことで、シンプルで一貫性のあるリクエスト標準化が円滑に行える。

HATEOAS

Hypermedia **A**s **T**he **E**ngine **O**f **A**pplication **S**tate
It means that hypertext should be used to find your way through the API.

完全にオンライン化されていてWebAPIで注文できるというすごいカフェを想定します。
客側から見たユースケースはこんな感じ
- 客はレジのサービスを呼び出して、注文を入力して支払い
- 自席で注文状況をチェック、出来上がっていたら取りに行く。

注文というエンティティと、[注文編集] [支払い] [受け取り] という(アプリケーション)状態によって上手く表現できそうです。

(RESTfulだけど)CRUDな設計

CURDな設計では、リソースをURLにマッピングします。それに対してCRUDするというイメージです。
- /order にPOST して注文を作成する
- /order/1234 にPUT : トッピングを追加
- /order/1234 にPUT:[支払い]に状態変更
- /order/1234 にPUT:[受取]に状態変更
- /order/1234 をGETして状況確認

RESTを意識するかぎり、状態に関する情報をセッション等に保持するのは避けたいので、注文エンティティに持たせることになるでしょう(アプリケーション状態からリソース状態に昇格させる)。そしてPOSTによって注文リソースを編集することによって、状態遷移を引き起こします。
この状態遷移のルールは、アプリケーションの中に隠されてます。
- 支払いをすると受け取り可能になる
- 支払いをする前に受取に状態変更しようとするとエラー
- 支払い済みものに、トッピングを追加しようとするとエラー
- ...

この例では、実はクライアント側にある種の前提を強いています。つまり状態遷移のモデルをクライアント開発側も把握している必要があります。

HATEOASな設計

HATEOAS は Hypermedia as the Engine of Application State の略です。状態マシンをハイパーメディアで表現するという意味ですね。
HATEOASな設計では、状態をURLにマッピングし、アクションをHTTPメソッドに、状態遷移をリンクにマッピングします。
[注文前の状態] /order
- POSTでcreate : 次の状態を返却[/order/1234]

[注文] /order/1234
- GETで注文内容確認
- PUTで注文編集 : 次の状態を返却[/order/payment/1234]

[支払い] /order/payment/1234
- PUTで支払い確定 : 次の状態を返却[/order/receive/1234]

[受け取り] /order/receive/1234
- GETしてチェック

状態遷移のルールは、URLとハイパーリンク、HTTPメソッドの制限によって表現できます
- 注文編集をした後に遷移する次の状態は、支払いの状態を意味するハイパーリンク(/order/payment/1234)を返却データに含めることで表現。
- 支払い前に受け取りできないのは、注文作成後にいきなり /order/recieve/1234 をGETすると404になることで表現。

- 支払い後に注文できないのは、支払い後に order/1234 のPUTが制限されることによって表現。(OPTIONSで確認可能)

HATEOASな設計ではクライアント側に必要な前提条件を減らすことができます。状態マシンが変更されても、それはURLとハイパーリンクの変更で表現されます。たとえば、[注文]と[支払い]の間に[スピードくじ抽選]という状態ができても、注文後に遷移する先が変更されるだけです。(完全に前提知識無しというのは無理でしょうが)

JSON format file

```
"name": "iPad",
        "links": [ {
            "rel": "self",
            "href": "http://localhost:8080/product/1"
        } ],
```

providing the links as shown above gives the client the information they need to navigate the service.

この場合、クライアントの役割は、HATEOASなAPIのブラウザになる。
結果として状態遷移とフロー構造を外在化・明示化することになる
結果として、Restful APIを介して、クライアントとサーバのより明確な構造的分離が達成できる

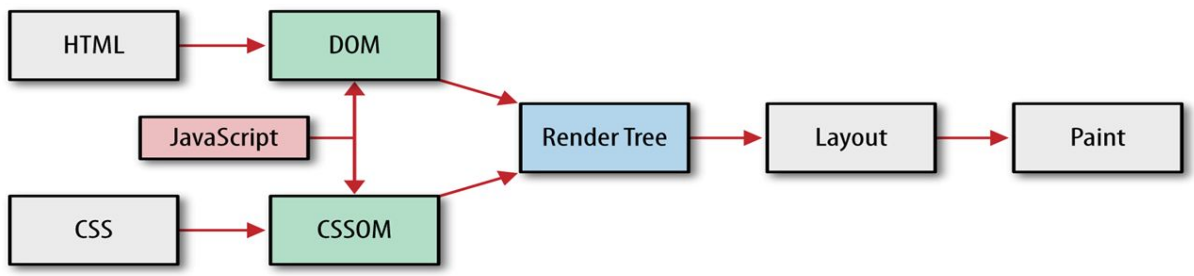
The way that the HATEOAS constraint decouples client and server enables the server functionality to evolve independently.

Question of client and server

1. What kind of data is sent from server to client?
    a. (Not protocol (HTTP), but content.)
2. What kind of data is sent from client to server?
    a. (Not protocol (HTTP), but content.)
3. Why have the kinds of data being sent changed over time?
    a. User experience
    b. Security
    c. Functionality
    d. Performance
    e. Reliability
4. How have the kinds of data being sent changed over time?
    a. Once upon a time: Static HTML, with data embedded into the HTML on the server side.

   b. ‘Application’ (of HTML, CSS, JavaScript) downloaded to the client; and then an exchange of data between client and server
5. How have the ways in which data is sent changed over time?
   a. Increasing use of AJAX, in which the application downloaded data behind the scenes, which means (for example) many HTTP GET requests.
   b. Change in the format of the data e.g. from XML to JSON. JSON is (much more) concise/efficient.
   c. Increasing use of micro-services i.e. pulling data from various sources
   d. Increasing use of CDNs

Document Object Model (DOM)



When a browser displays a document, it must combine the document's content with its style information. It processes the document in two stages:

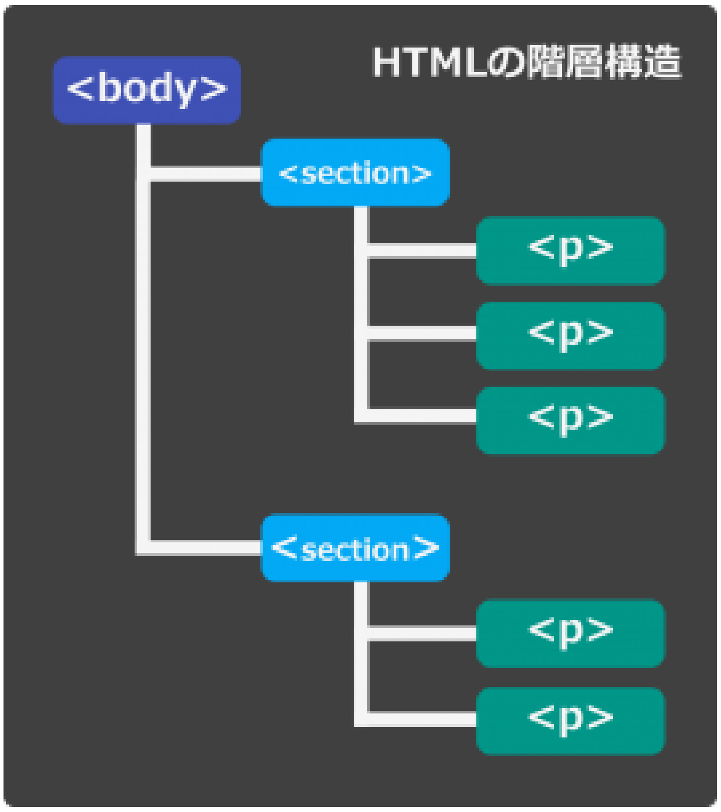
- The browser converts HTML and CSS into the DOM(Document Object Model). The DOM represents the document in the computer's memory. It combines the document's content with its style.
- The browser displays the contents of the DOM.

A DOM has a tree-like structure. Each element, attribute and piece of text in the markup language becomes a DOM node in the tree structure. The nodes are defined by their relationship to other DOM nodes. Some elements are parents of child nodes, and child nodes have siblings.
Understanding the DOM helps you design, debug and maintain your CSS because the DOM is where your CSS and the document's content meetup.

The DOM defines a standard for accessing documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

DOMとは、HTML の各要素にアクセスする仕組み！
JavaScript や CSS を使い要素の値を直接操作する



<body>を頂点として、下にいくつかの<section>と、そのさらに下にいくつかの<p>で構成されている。これはHTMLで階層構造を構築した場合の一例だ。この階層構造を定義しているものこそがDOMと呼ばれている仕組みになる。

Being stateless for HTTP
A **stateless protocol** is a communications protocol in which no information is retained by either sender or receiver. It does not require the server to retain session information or status about each communications partner for the duration of multiple requests.
The stateless design simplifies the server design because there is no need to dynamically allocate storage to deal with conversations in progress.

HTTP is a “stateless” protocol

- Means credentials have to go with every request

- Should use SSl for everything requiring authentication.

Cookies

A small piece of data initially sent by the server to the client.
Used to maitain state information.

- e.g. items in a shopping basket
- e.g. browser activity such as a ‘path’ trough a resistration process.

Web application security

OWASP
Open Web Application Security Project

The OWASP Top 10 project

1. **Injection** - Injection flaws, such as SQL, OS, XXE, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
   How to prevent injection attacks
   To prevent server-side js injection attacks:
   - Validate user inpus on server side before processing
   - Do not use the ‘eval()’ function to parse user inputs. Avoid using other commands with similar effect, such as ‘setTimeOut ()’, ‘setInterval ()’, and ‘Function ()’.
   - Form parsing JSON input, instead of using eval (), use a safer alternative such as JSON.parse (). For type conversions use type relatedparseXXX () methods.
   - include “use strict”; at the top of each source file.
2. **Broken Authentication and Session Management** - Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users’ identities (temporarily or permanently).
   How to prevent broken authentication vulnerabilities
   - **Protecting user credentials:** use hash and salt to make password hard to steal
   - **Session timeout and protecting cookies in transit**

- **Password guessing attacks:** use more than 8 letters, combination of numerical and strings password, and not make failure responses indicate which part of the authentication data was incorrect.

3. **Cross-Site Scripting (XSS) -** XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user supplied data using a browser API that can create JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
    a. **Reflected XSS :** The malicious data is echoed back by the server in an immediate response to an HTTP request from the victim.
    b. **Stored XSS :** The malicious data is stored on the server or on browser (using HTML5 local storage, for example), and later is embedded in an HTML page provided to the victim.

    How to prvent XSS attacks
    - **Input validation and sanitization:** Input validation and data sanitization are the first line of defense against untrusted data. Apply white list validation wherever possible.
    - **Output encoding for correct context:** When a browser is rendering HTML and any other associated content like CSS, JavaScript etc., it follows different rendering rules for each context. Hence Context-sensitive output encoding is absolutely critical for mitigating risk of XSS.
    - **HTTPOnly cookie flag:** Preventing all XSS flaws in an application is hard. To help mitigate the impact of an XSS flaw on your site, set the HTTPOnly flag on session cookie and any custom cookies that are not required to be accessed by JavaScript.
    - **Implement Content Security Policy (CSP):** CSP is a browser side mechanism which allows creating whitelists for client side resources used by the web application, e.g. JavaScript, CSS, images, etc. CSP via special HTTP header instructs the browser to only execute or render resources from those sources. For example, the CSP header below allows content only from example site's own domain (mydomain.com) and all its sub domains.
    - **Content-Security-Policy: default-src 'self' *.mydomain.com**
    - **Apply encoding on both client and server side:** It is essential to apply encoding on both client and server side to mitigate DOM based XSS attack, in which untrusted data never leaves the browser.

4. **Broken Access Control -** Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access

unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

5. **Security Misconfiguration** - Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, platform, etc. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.
6. **Sensitive Data Exposure** - Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.
7. **Insufficient Attack Protection** - The majority of applications and APIs lack the basic ability to detect, prevent, and respond to both manual and automated attacks. Attack protection goes far beyond basic input validation and involves automatically detecting, logging, responding, and even blocking exploit attempts. Application owners also need to be able to deploy patches quickly to protect against attacks.
8. **Cross-Site Request Forgery (CSRF)** - A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. Such an attack allows the attacker to force a victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.
9. **Using Components with Known Vulnerabilities** - Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.
10. **Underprotected APIs** - Modern applications often involve rich client applications and APIs, such as JavaScript in the browser and mobile apps, that connect to an API of some kind (SOAP/XML, REST/JSON, rPC, GWT, etc.). These APIs are often unprotected and contain numerous vulnerabilities.

ACID
is a set of properties of database transactions intended to guarantee validity even in the event of errors, power failures, etc.

Atomicity
Atomicity requires that each transaction be "all or nothing": if one part of the transaction fails, then the entire transaction fails, and the database state is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors and crashes. To the outside world, a committed transaction appears (by its effects on the database) to be indivisible ("atomic"), and an aborted transaction does not happen.

口座Aから口座Bに対し1万円送金する場合を考えたとき、送金操作は次の2操作によって行われる。

1. 口座Aの残高から1万円を引く
2. 口座Bの残高に1万円を加える

原子性が保証されるとは、上の操作1、2が全て行われるか、あるいは全く行われないことを指す。どちらか片方だけが実行された場合、銀行全体の預金残高に矛盾が発生してしまう。

Consistency
The consistency property ensures that any transaction will bring the database from one valid state to another. Any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof.

日本語では一貫性あるいは整合性とも呼ばれる。トランザクション開始と終了時にあらかじめ与えられた整合性を満たすことを保証する性質を指す。すなわち、データベースのルール、つまり整合性条件を満たさない状態を起こすようなトランザクションは実行が中断される。

預金残高を例にすると、その値は一般的に0あるいは正の値を取る条件を満たす必要がある。よって、口座Aから送金を行うとき、その前後でAの口座残高が負になるような額は送金できないようにする。このようなルールを保証するのが一貫性の役割である。

Isolation

The isolation property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed sequentially. Providing isolation is the main goal of concurrency control. Depending on the concurrency control method the effects of an incomplete transaction might not even be visible to another transaction.

トランザクション中に行われる操作の過程が他の操作から隠蔽されることを指し、日本語では分離性、独立性または隔離性ともいう。より形式的には、独立性とはトランザクション履歴が直列化されていることと言える。この性質と性能はトレードオフの関係にあるため、一般的にはこの性質の一部を緩和して実装される場合が多い。

預金残高の例では、残高100万円の口座Aから残高200万円の口座Bに1万円送金する場合の操作が

1. 口座Aの残高から1万円を引く
2. 口座Bの残高に1万円を加える

の順序で行われたとする。取りうる内部状態は、

時点	口座A	口座B
送金前	100万円	200万円
実行中	99万円	200万円
送金後	99万円	201万円

の3つになるが、外部からは**送金前**と**送金後**のいずれかの状態しか観測できない。

Durability

永続性あるいは持続性と呼ばれる。トランザクション操作の完了通知をユーザが受けた時点で、その操作は永続的となり、結果が失われないことを指す。これはシステム障害に耐えるということであり、DBMSは整合性制約をチェック済みでありトランザクションを中止してはいけないということである。多くのデータベース実装では、トランザクション操作を永続性記憶装置上にログとして記録し、システムに異常が発生したらそのログを用いて異常発生前の状態まで復旧することで永続性を実現している。

The durability property ensures that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors. In a relational database, for instance, once a group of SQL statements execute, the results need to be stored permanently (even if the database crashes immediately thereafter). To defend against power loss, transactions (or their effects) must be recorded in a non-volatile memory.

Write-ahead logging

In computer science, write-ahead logging (WAL) is a family of techniques for providing atomicity and durability (two of the ACID properties) in database systems.

In a system using WAL, all modifications are written to a log before they are applied. Usually both redo and undo information is stored in the log.

The purpose of this can be illustrated by an example. Imagine a program that is in the middle of performing some operation when the machine it is running on loses power. Upon restart, that program might well need to know whether the operation it was performing succeeded, half-succeeded, or failed. If a write-ahead log is used, the program can check this log and compare what it was supposed to be doing when it unexpectedly lost power to what was actually done. On the basis of this comparison, the program could decide to undo what it had started, complete what it had started, or keep things as they are.

CAP Theorem

ノード間のデータ複製において、同時に次の3つの保証を提供することはできない。

- ***一貫性*** (**C**onsistency)
    - すべてのデータ読み込みにおいて、最新の書き込みデータもしくはエラーのどちらかを受け取る。
    - Every read receives the most recent write or an error
- ***可用性*** (**A**vailability)
    - ノード障害により生存ノードの機能性は損なわれない。つまり、ダウンしていないノードが常に応答を返す。単一障害点が存在しないことが必要。
    - Every request receives a (non-error) response – without guarantee that it contains the most recent write
- ***分断耐性*** (**P**artition-tolerance)
    - システムは任意の通信障害などによるメッセージ損失に対し、継続して動作を行う。通信可能なサーバーが複数のグループに分断されるケース（ネットワーク分断）を指し、1つのハブに全てのサーバーがつながっている場合は、これは発生しない。ただし、そのような単一障害点のあるネットワーク設計は可用性が成立しない。
    - The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

この定理によると、分散システムはこの3つの保証のうち、同時に2つの保証を満たすことはできるが、同時に全てを満たすことはできない。可用性を成立させるには単一障害点をなくさないといけないが、すると、ネットワーク分断が発生した際、システムがバラバラに分裂してしまい、単一障害点があればそこを基準に一貫した応答ができるが、単一障害点をなくしてしまうとシステムの応答の一貫性が成立できなくなるという定理。

But New formulation

Normal (Network up) => has to choose Latency or Consistency

Partion (Network Down) => Availability or Consistency

If sacrifice Availability

consistent but not available for some people

If sacrifice Consistency

Can’t update instantly but eventualy => **Eventual Consistency**

Process flow of partition resolve

1. **Normal**
2. **Detect partition**
3. **Availability or Consistency**
4. **Resolve**

Availability を優先すると Eventual Consistency

Consistency を優先すると今は使えない等のメッセージほ機械上に表示させ使用を制限する。

ATM Example

Choose Availability!!

客に対する利便性を考慮して

but have to becareful for withdrawal.

There 2 options for solution for withdraul

1 is make it have limitation to withdrawl

another one is phone to police etc.

Command and Query Responsibility Segregation (CQRS)

write data store と read data store があり、incoming data goes into write data store and it updates recent data into read data store so clients only need to read new data and no need to look up whole data to find out the specific data.

Single Page Application (SPA) Single page apps are distinguished by their ability to redraw any part of the UI without requiring a server roundtrip to retrieve HTML. This is achieved by separating the data from the presentation of data by having a model layer that handles data and a view layer that reads from the models.

Advantage

1. FaceBook のようにChat window を表示したままContents の表示を変更する等が出来るようになる。
2. HTML全体をServerから取得しなくても済むため、より高速で表示できる。

Disadvantage

1. ページ毎のURLの割り当て等browserに任せていた処理を実装する必要がある！
2. JavaScriptのコード量が増えるため、今までserver側で行っていたHTMLの生成をbrowserでやらないといけないため初期ローディングに時間がかかる。

How do you get data from the server without refreshing the page?

XHR / AJAX

XMLHttpRequest

クライアントとサーバーの間でデータを伝送するための機能をクライアント側で提供する API です。ページ全体を再読み込みすることなく、URL からデータを読み出す簡単な方法を提供します。この API によって、ユーザの作業を中断させることなく Web ページの一部を更新することができます

Despite the name, XHR can be used with protocols other than HTTP and data can be in the form of not only XML, but also JSON, HTML or plain text.

Templating

JavaScript templating refers to the client side data binding method implemented with the JavaScript language.

- A way of separating the HTML structure from the content
- Templating introduces additional (but minimal) syntax
  - e.g. {{ . . . }}
- Thereare inline and external templates
- Options to include the template in the HTML or in the JavaScript

```
<script id="template" type="x-tmpl-mustache">
  <p>Use the <strong>{{power}}</strong>, {{name}}!</p>
</script>
//Grab the inline template
var template = document.getElementById('template').innerHTML;
//Parse it (optional, only necessary if template is to be used again)
Mustache.parse(template);
//Render the data into the template
var rendered = Mustache.render(template, {name: "Luke", power: "force"});
//Overwrite the contents of #target with the rendered HTML
document.getElementById('target').innerHTML = rendered;
```

Separating data from contebt and presentation

- **Templating (NOT to be confused with HTML5 <template>)** helps us to separate data from content & presentation in terms of rendering to the screen.
- **XHR** helps us to separate data from content and presentation
- Use XHR to send and retrieve data independently of content and presentation.

Cross Origin Resource Sharing

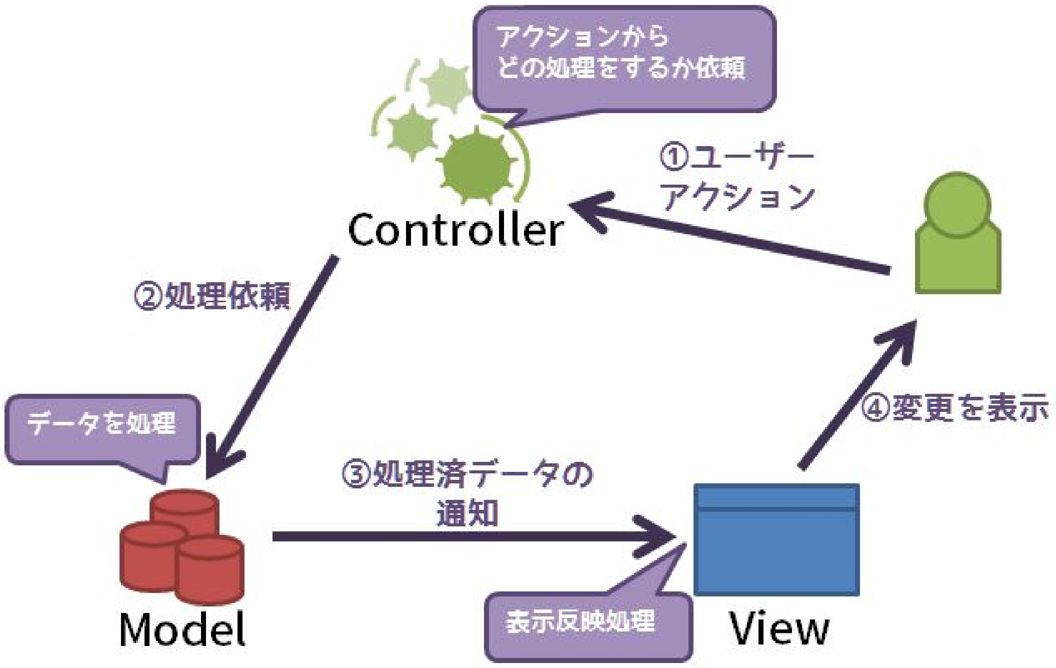
- it defines a way in which a browser and a server can interact to determine whether or not it is safe to allow the cross-origin request.
- The CORS standard describes HTTP headers which provide browsers and servers a way to request remote URLs only when they have permission.

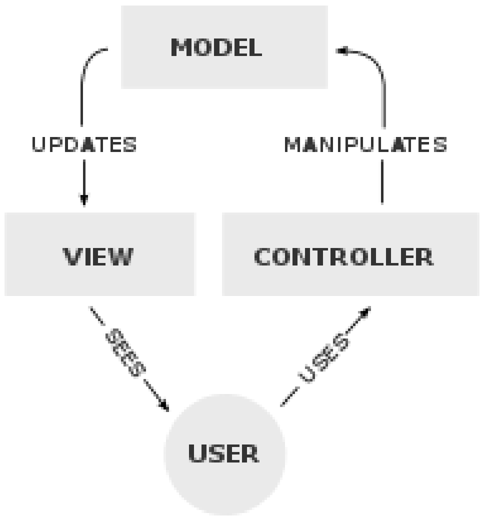
- It is generally the browser’s responsibility to support these headers and honour the restrictions they impose. **not the application’s responsibility to prevent the application doing something.**

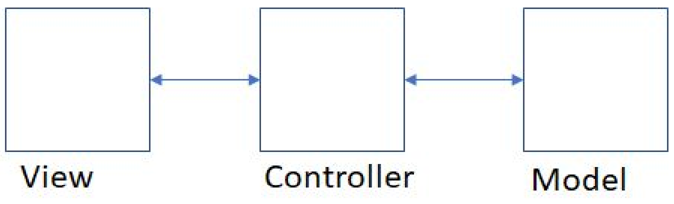
Model View Controller (MVC)

データを効率的に使える！！

Traditionally used for desktop GUIs, this architecture has become popular for designing web applications.







Model

Application data, logic and function

Applicationが扱う領域のデータと手続き（ショッピングの合計額や送料を計算するなど）を表現する要素。また、データの変更をViewに通知する。

- Central component of the pattern
- Expresses the application’s behaviour in terms of the problem domain, independent of the user interface.
- Directly manages the **data, logic and rules of the application.**

View

Graph or table

Modelのdataを取り出してUserが見るのに適した形で表現する要素。すなわち、UIへの出力を担当する。例えば、Web ApplicationではHTML文書を生成して動的にデータを表示するためのコードなどにある。

- Any output representation of information, such as a chart or a diagram, or even text.
- Multiple views of the same information are possible e.g. a bar chart and tabular view of the same model.

Controller

入力を受け取りmodelとviewへの命令に変換する

UserからのInputをmodelへのmessageへと変換してmodelに伝える要素。すなわちUIからの入力を担当する。modelに変更を引き起こす場合もあるが、直接に描画を行ったり、modelの内部データを直接操作したりはしない。

- Accepts input and converts it to commands for the model or view.

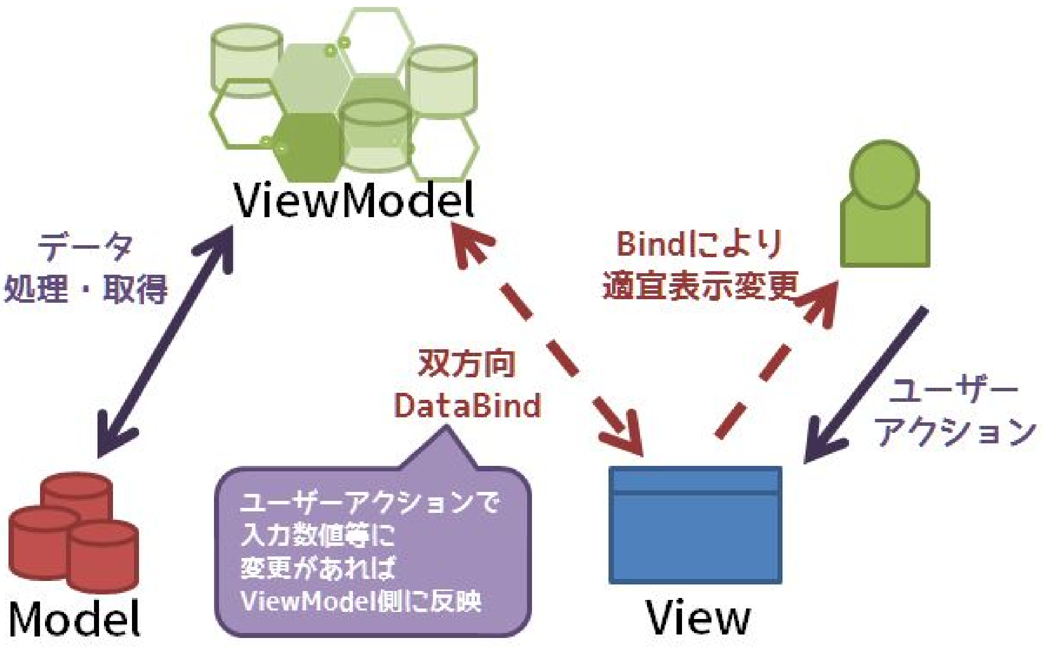
MVCのシナリオ

1. Controllerが入力機器（マウスやキーボードなど）を監視する。
2. ユーザが入力機器に入力を与える。

3. ControllerがUserのアクションに応じてModelのメソッドを呼ぶ。その結果Modelのデータが書き換えられる場合もある。
4. Modelが変更された場合、自身が変更された旨をViewに対して通知する。
5. ViewはModelから関連するデータを取得し、出力を更新する。主に画面に図形を描画する。

- **Model**
  - Stores data that is retrieved according to commands from the controller and displayed in the view.
- **View**
  - Generates new output to the user based on changes in the model
- **Controller**
  - Send commands to the model to update the model’s state (e.g. editing a document). It can also send **commands to its associated view to change the view’s presentation of the model** (e.g. scrolling through a document, movement of document).

Model View ViewModel (MVVM)



双方向データバインディング（双方向DataBind）という形でViewとViewModelを結びつけています。

双方向データバインディング

データバインディングとは文字の通りデータ(Data)を結びつける(Bind)する機能。

UIとデータオブジェクトを結びつけて、同一のデータをやりとりできるようにしてUIが操作がされるとデータ側も書き換えるような仕組み。

また、双方向データバインディングの場合はデータが書き換えられてもUIに反映、UI側の情報が書き換えられても反映というような相互でデータの更新をやりとりするようなイメージ。

Model

- Either to a domain model, which represents real state content (an object approach), or to a data access layer, which represents content (a data-centric approach).
- アプリケーションのドメイン（問題領域）を担う、そのアプリケーションが扱う領域のデータと手続き（ショッピングの合計額や送料を計算するなど）を表現する要素。
- 多くのアプリケーションではデータの格納に永続的な記憶の仕組み（データベースなど）が使われていたり、サーバが別途存在するアプリケーションではサーバ側との通信ロジックなどが含まれている。
- MVVMの概念ではMVCの概念と同様に、データの（UI以外の）入出力は取り扱わないので、強いて言うならばそれらはModelの中に隠蔽されると考えられる

View

- The view is the structure, layout, and appearance of what a user sees on the screen (HTML & CSS)
- アプリケーションの扱うデータをユーザーが見るのに適した形で表示し、ユーザーからの入力を受け取る要素。すなわちUIへの入力とUIからの出力を担当する。
- MVVMにおけるViewは、後述するViewModelに含まれたデータをデータバインディング機構のようなものを通じて自動的に描画するだけで自身の役割を果たす

ViewModel

- An abstraction of the view exposing public properties commands. Instead of the controller of the MVC pattern, or the presenter of the MVP pattern, MVVM has a binder.

- In the ViewModel, the binder mediates communication between the view and the data binder. The view model has been described as a state of the data in the model.
- Viewを描画するための状態の保持と、Viewから受け取った入力を適切な形に変換してModelに伝達する役目を持つ。すなわちViewとModelの間の情報の伝達と、Viewのための状態保持のみを役割とする要素
- **Model**から取得したデータをバインディングできるようにデータを定義し表示と連動してデータが変更され、またデータに直接操作が必要な場合は**Model**へ操作を依頼するものとして存在するのが**ViewModel**。

Binder

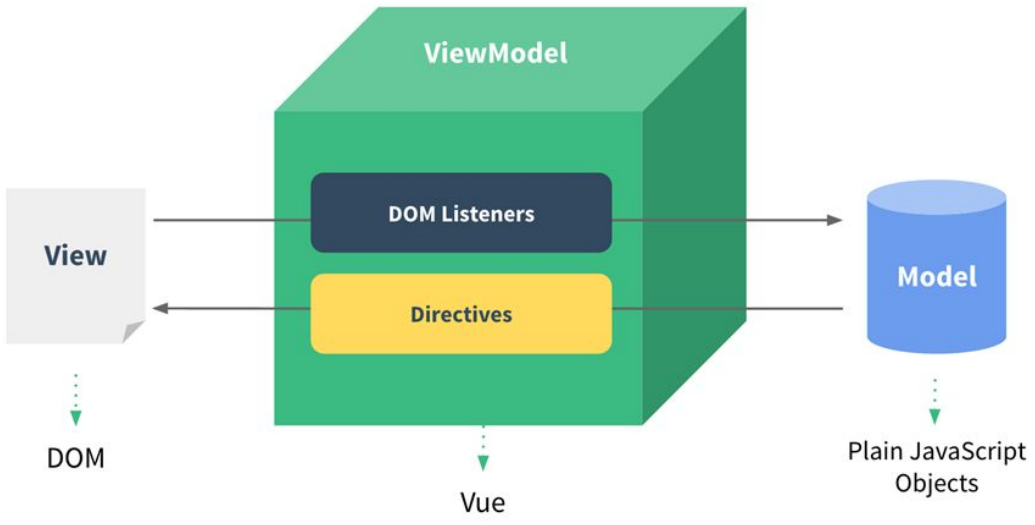
- Declarative data and command-binding are implicit in the MVVM pattern.

How does this apply to Vue.js?

Technically, Vue.js is focused on the **ViewModel** layer of the MVVM pattern. It connects the **View** and the **Model** via two way data bindings. Actual DOM manipulations and output formatting are abstracted away into **Directives** and **Filters**.

Vue.js

- Vue allows you to link data and the DOM
- Once linked, things are then reactive
    - User changes the DOM (e.g. enters information), the data is updated.
    - JavaScript changes the data, the DOM is updated.



ViewModel

```
let vm = new Vue({
    /* options */
})
```

- The ViewModel is an ‘object’ (not an object in the OO sense) that syncs the Model and the View.
- In Vue.js every Vue instance is a ViewModel.
- ViewModels are instantiated with the Vue constructor or its sub-classes.

View

```
vm.$el // The View
```

- The actual DOM that is managed by Vue instances.
- Vue.js uses DOM-based templating.
- Each Vue instance is associated with a corresponding DOM element.
    - Multiple Vue instances
    - Multiple MVVMs in a Vue application
- When a Vue instance is created, it recursively walks all child nodes of its root element while setting up the necessary data bindings.
- After the View is compiled, it becomes reactive to data changes.
- When using Vue.js, yourarely have to touch the DOM yourself except in custom directives.
- View updates will be automatically triggered when the data changes.
- These view updates are highly granular with the precision down to a textNode.
- They are also batched and executed asynchronously for greater performance.
    - Virtual DOM

Model

```
vm.$data // The Model
```

```
let vm = new Vue({
    ...
    data:...
    ...
})
```

- In Vue, models are Plain Old JavaScript Objects
- Model is used as data in a Vue instance
    - Declared in the data option

Directives

Prefixed HTML attributes that tell Vue.js to do something about a DOM element.

```
<div v-text="message"></div>
```

Here the div element has a `v-text` directive with the value `message`. This tells Vue.js to keep the div’s textContent in sync with the Vue instance’s `message` property.

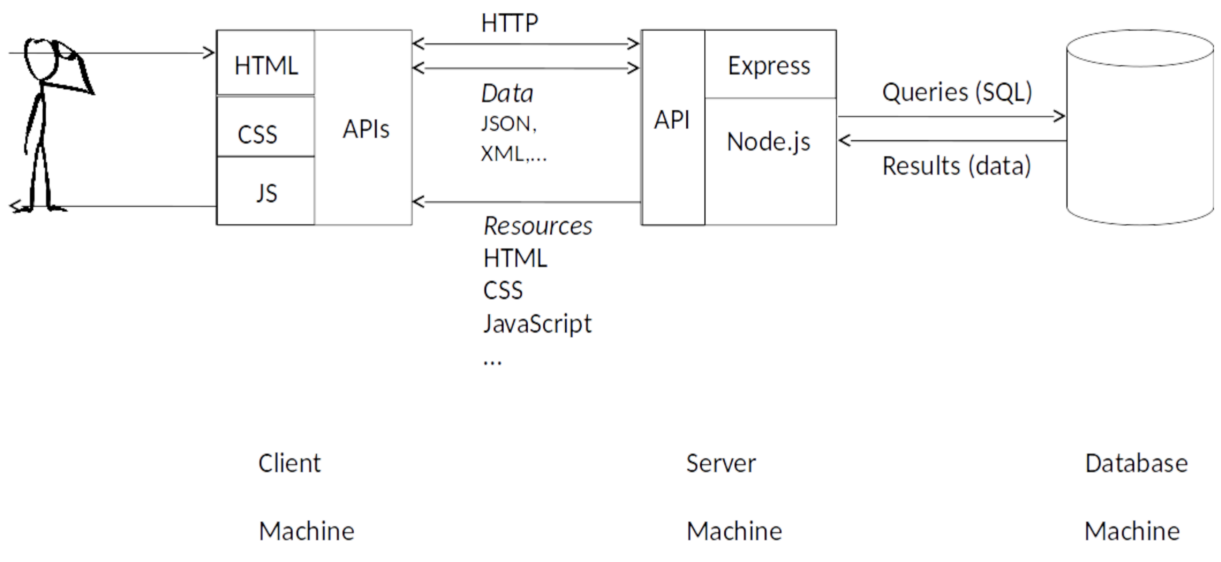
Filters

Filters are functions used to process the raw values before updating the View. They are denoted by a “pipe” inside directives or bindings:

```
<div>{{message | capitalize}}</div>
```

Now before the div’s textContent is updated, the `message` value will first be passed through the `capitalize` function.

Past Exam
2016
Explain how a web client and web server communicate when the client makes a GET request to, and receives a response from, the server for a resource, and where that resource contains dynamically generated data. Your explanation should include components such as the client, communication protocol/s, web server, server-side scripting, database server, and data format. You do not need to write code for your answer however a stronger answer would include technical detail on the request-response cycle. You may include one or more diagrams in your answer.



API … The things that allows machine to communicate
SSL … Secure sockets layer