Week 5: Controllers - Business logic

L5.1: *MVC Origins*

Controllers

- Controller is responsible for handling user interactions and acting as an intermediary between the **Model** and the **View**.
- It receives input from the user through the **View** and updates the **Model** accordingly, orchestrating the flow of data and business logic.
- The controller communicates changes in the **Model** to the **View**, ensuring that the user interface reflects the most up-to-date and responding to user actions appropriately.

Action

1. **Receiving User Input**: The controller receives user input and triggers appropriate actions based on the input.
2. **Updating the Model**: It communicates the model to update or retrieve data, implmenting the application's business logic.
3. **Updating the View**: The controller also interacts with the view to update the user interface and display the relevant data to the user.

Applicability

- It is originally designed for GUI applications.
- For web applications, the server doesn't maintain state of client, hence client is pure front-end to the user.

1. **Separation of Concerns**: The controller promotes a clear separation between user interactions (handled by the controller), data and business logic (contained in the model), and presentation (handled by the view). This separation allows for a modular and organized codebase, making the application easier to understand, maintain, and scale.
2. **User Interaction Management**: The controller is responsible for capturing and managing user input, ensuring that it is appropriately processed and validated. It facilitates the interaction between the user and the application, orchestrating the flow of data and actions based on user input.
3. **Flexibility and Reusability**: The controller's decoupling from the model and view allows for greater flexibility and reusability. Different views can interact with the same controller and model, enabling the development of multiple user interfaces without changing the underlying business logic. Similarly, the same controller can be utilized across different applications, enhancing code reuse.

L5.2: *Requests and Responses*

Request

- A request is made by the client to the server, asking for specific information or to perform an action.
- It typically includes a **URL** that specifies the resource or endpoint the client wants to access, as well as additional data like form data, query parameters, or request headers.
- Some common request methods are **GET**, **POST** and **DELETE**.

Response

- A response is sent by the server to the client, containing the requested information or confirming that the requested action has been performed.
- Once the server receives a request, it processes the request and sends back a response to the client.
- The response typically contains the requrested data with a status code indicating the success or failure of the request.
- The response may also include response headers that provide additional information about the data or how it should be handled by the client.

Example:

1. API request & response

Request: `GET https://www.example.com/api/posts?category=technology&limit=2`

```
GET /api/posts HTTP/1.1
Host: www.example.com
```

- In this **GET** request, the client it requesting data from the server using the **GET** method.
- The base URL is `https://example.com`, and the endpoint is `/api/posts`.
- The query parameters are appended to the URL after the `?` character, with each parameter consisting of a key-value pair separated by the `=` character.
  - `category=technology`: This is a query parameter with the name *`category`* and the value *`technology`*.
  - `limit=2`: This is another query parameter with the name *`limit`* and the value *`2`*.
- The server will use these parameters to filter the results and respond with a list of posts from the "technology" category, limited to 10 posts.

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 123,
        "title": "Sample Post",
        "content": "This is the content of the post."
    },
    {
        "id": 123,
        "title": "Sample Post",
        "content": "This is the content of the post."
    }
]
```

2. HTML webpage request & response

Request: `GET https://www.example.com/blog`

```
GET /index.html HTTP/1.1
Host: www.example.com
```

- In this **GET** request, the client is requesting an HTML webpage from the server using the **GET** method.
- The base URL is `https://www.example.com`, and the endpoint is `/blog`.
- The server will respond with the HTML content of the webpage.
- The default file name is `index.html`, so the server will look for a file named `index.html` in the `/blog` directory.

Response:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 100
```

```
<html>
    <head>
        <title>My Website</title>
    </head>
    <body>
        This is my website.
    </body>
</html>
```

L5.3: *CRUD*

- ***CRUD** stands for **Create**, **Read**, **Update**, and **Delete***.
- It is a set of four basic operations that are used to manage data in a database.
- CRUD is a very common set of operations that are used in many different types of software applications.
- It is a fundamental part of database management, and is essential for any application that needs to store and manage data.

CREATE

- Create is used to add new data to the database.
- The data must not exists, be valid and meet the requirements of the database.
- **Example**:
  - Create a new product by entering the product name, description, price, and other relevant information.

READ

- Read is used to retrieve data from the database.
- The data must exists, be accessible and the user must have the necessary permissions to view it.
- **Example**:
  - Read the product database to retrieve information about a specific product, such as the product name, description, price, and stock level.

UPDATE

- Update is used to change existing data in the database.
- The data must exists, be valid and meet the requirements of the database. The user must have the necessary permissions to update the data.
- **Example**:
  - Update the product database to change the product name, description, price, or other relevant information.

DELETE

- Delete is used to remove data from the database.
- The data must exists, be accessible and the user must have the necessary permissions to delete it.
- **Example**:
  - Delete a product from the product database.

API

- API stands for **Application Programming Interface**.
- It is a set of definitions and protocols that allow two applications to communicate with each other.
- APIs are used to share data and functionality between applications.
- APIs provide a standardized interface, abstracting the underlying complexity of systems.
- APIs follow a request-response model, where one application sends requests to another to access data or

perform actions.

- There are many different types of APIs, such as **Web APIs**, **Library APIs**, **Database APIs** and **Hardware APIs**.
  - **Web APIs** includes **REST APIs**, **RPC APIs**, **GraphQL APIs**, etc...

Example:

A waiter in a restaurant

- The waiter is the intermediary between the customer and the kitchen. The waiter takes the customer's order and then relays it to the kitchen. The kitchen then prepares the food and the waiter delivers it to the customer.
- Here's how it works:
  1. The customer makes a request to the waiter.
  2. The waiter receives the request and relays it to the kitchen.
  3. The kitchen receives the request and prepares the food.
  4. The kitchen delivers the food to the waiter.
  5. The waiter delivers the food to the customer.

API for CRUD

- An API for CRUD is a set of endpoints that allow users to perform CRUD operations on a database.
- These endpoints are typically exposed via HTTP requests, and they use the HTTP methods `GET`, `POST`, `PUT`, and `DELETE` to represent the different CRUD operations.

GET

- The `GET` is used to retrieve data from the database.
- **Example**:
  - `GET /products` - This endpoint would return a list of all products in the database.

POST

- The `POST` is used to create new data in the database.
- **Example**:
  - `POST /products` - This endpoint would create a new product in the database.

PUT

- The `PUT` is used to update existing data in the database.
- **Example**:
  - `PUT /products/1` - This endpoint would update the product with the ID of 1.

DELETE

- The `DELETE` is used to delete data from the database.
- **Example**:
  - `DELETE /products/1` - This endpoint would delete the product with the ID of 1.

REST API

- In our course, we will be focusing on **REST APIs**.
- `REST API` stands for **Representational State Transfer Application Programming Interface**.
- It is a style of API design that uses HTTP requests to access and manipulate resources.
- REST APIs are based on the following principles:
  - Resources are identified by URIs (Uniform Reosucre Identifiers).
  - Reosurces are represented in a format that is easy ot understand and process, such as JSON or XML.
  - Operations on resources are performed using HTTP methods, such as `GET`, `POST`, `PUT`, and `DELETE`.
  - Links are used ot represent relationship between resources.

Examples:

- The **GitHub API** allows you to access and programmatically interact with GitHub data.
- The **Google Maps API** allows you to embed maps in your website or application.
- The **Stripe API** allows you to accept payments in your website or application.

L5.4: *Group Actions by Controller*

- In larger web applications, there are multiple functionalities and actions that need to be performed.
- Grouping actions means organizing related actions together based on their functionality or purpose.
- Grouping helps in better code organization, readability and maintainability.

Example: *A controller for managing a blog using flask*

Verb	URI	Action	Route name
`GET`	`/blog`	List all blog posts	`blog_list`
`GET`	`/blog/<int:post_id>`	Get a blog post by ID	`blog_get`
`POST`	`/blog/create`	Create a new blog post	`blog_create`
`PUT`	`/blog/<int:post_id>`	Update a blog post	`blog_update`
`DELETE`	`/blog/<int:post_id>`	Delete a blog post	`blog_delete`
`GET`	`/blog/search`	Search for blog posts	`blog_search`

Rules of Thumb

1. Should be possible to change views without the model ever knowing
2. Should be possible to change underlying storage of model without views every knowing.

3. Controllers/Actions should generally NEVER talk to a database directly. It should always go through the model.

L5.5: *Routes and Controllers*

Routing

- It refers to the process of how incoming requests in a web application are handled and mapped to speific actions or resources.
- Or, you can say it is the process of mapping URLs to specific pages or components.
- When a user enters a URL in their browser, the web application's routing system will determine which page or component to load.

While using flask in Python, we do routing using decorators.

Decorators

- A decorator is a function that takes another function as an argument and returns a new function.
- It allows us to wrap one function inside another to add pre-processing, post-processing, or other additional functionalities.
- Decorators are defined using the @ symbol.

Example:

```
def my_decorator(func):
    def wrapper():
        print("Before function call")
        func()
        print("After function call")
    return wrapper

# Applying decorator to a function
@my_decorator
def greet():
    print("Hello")

greet()
```

Output:

```
Before function call
Hello
After function call
```

Usecases:

- There are many usecases of decorators, some of them are:
  - Logging
  - Caching and memoization
  - Validation
  - Timing
  - Rate Limiting
  - Error handling and a lot more..

Routing in Flask

- In flask, we use decorators to define routes.

```
from flask import Flask

app = Flask(__name__)

@app.route('/') # By default, method is GET
def index():
    return 'Hello World!'

@app.route("/create", methods=["POST"])
def create():
    # creates a new blog post using post method to fetch data
    return ...

@app.route("/<int:blog_id>", methods=["GET"])
def read():
    # fetches a blog post using GET method
    return ...

@app.route("/update/<int:blog_id>", methods=["GET", "POST"])
def update():
    # update the blog post, can use GET or POST method, based on the usecase
    return ...

def delete():
    # delete the blog post using DELETE method
    return ...

# another way of using decorators
@app.route("/delete/<int:blog_id>", methods=["DELETE"])(delete)
```

Screencasts

5.1: *How to create relational database using sqlite?* 🔗

5.2: ***How to query database usign SQLAlchemy?*** 🔗

5.3: ***Introduction to Flask SQLAlchemy - Part 1*** 🔗

5.4: ***Introduction to Flask SQLAlchemy - Part 2*** 🔗

Check SQL Alchemy in Flask here 🔗

5.5 ***Structure of a Flask App*** 🔗

Check Flask Project layout here 🔗

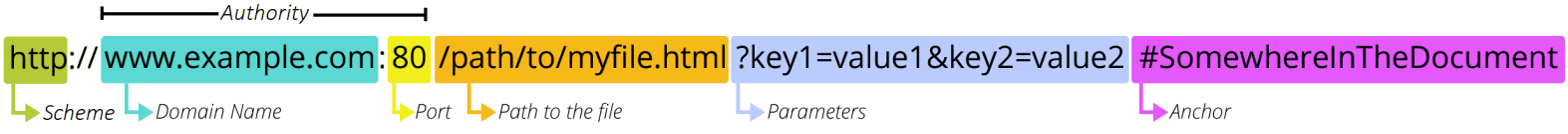
Additional material

Flask documentation 🔗

My Personal decorator notes

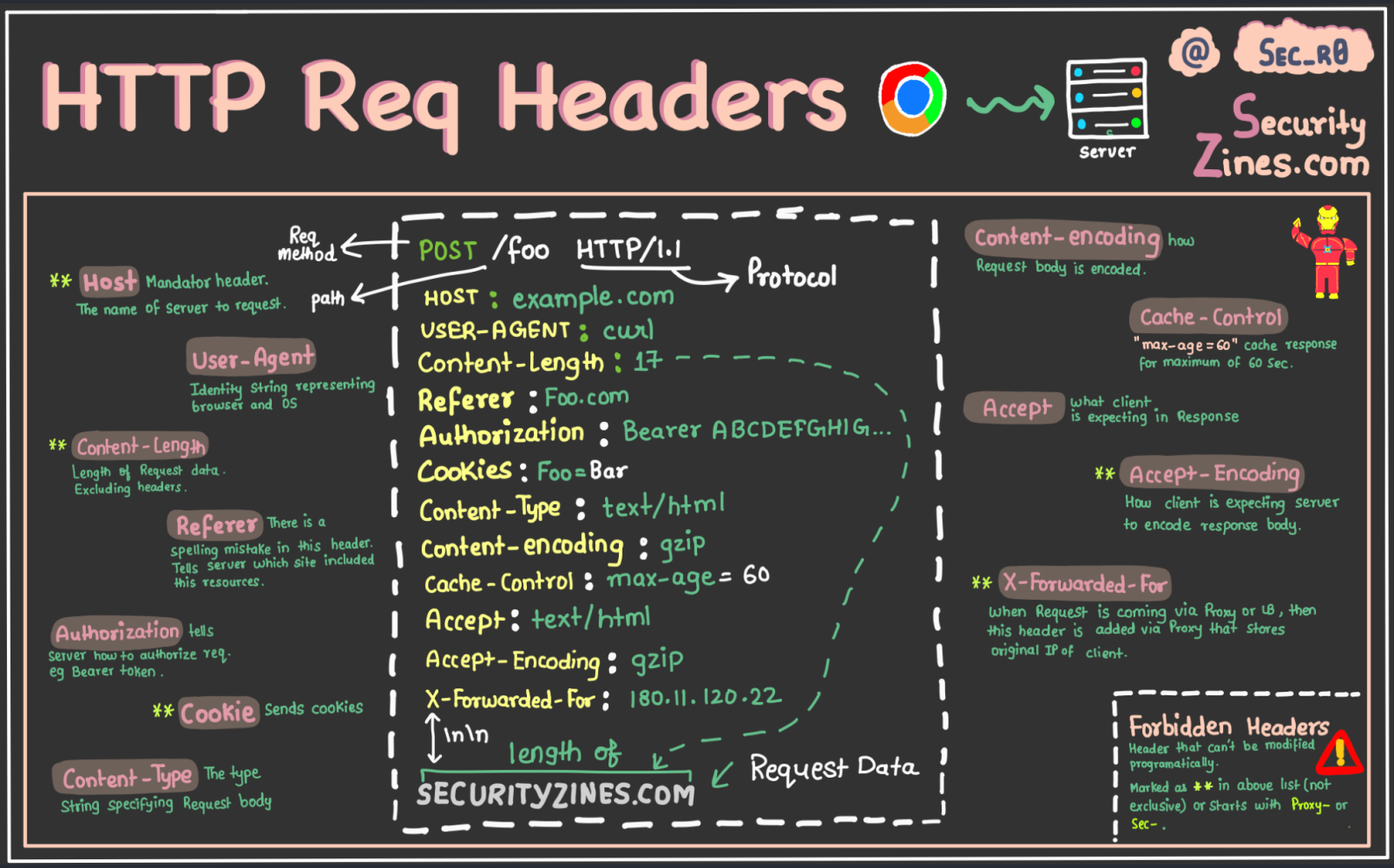
```
"""
Decorators:
It is a function that takes another function as an arugment,
adds some other functionality/features and returns another function,
without altering the code of passed function.

After decorating the function with a decorator,
it's attributes replaced by wrapper() or any inner() function,
as wrapper() is returning from the decorator.
Like name of function and docstring of function

__name__ , __doc__

So, to mantain that, we apply @wraps decorator from functools library to mantain the attributes
of original decorated function by passing the function name in @wraps() decorator,
and wrapping it in the wrapper of the decorator.

def decorator(func):

    @wraps(func)
    def wrapper(*args, **kwargs):
        '''A wrapper of the decorator'''
        return func(*args, **kwargs)

    return wrapper

@decorator
def hello(name):
    '''Greet the user with their name.'''
    print(f"Hello, {name}")

>>> print(hello.__name__)
>>> print(hello.__doc__)

BEFORE APPLYING @wraps(func)
Output:
wrapper
A wrapper of the decorator

AFTER APPLYING @wraps(func)
Output:
hello
Greet the user with their name.
"""
```

Anatomy of a URL



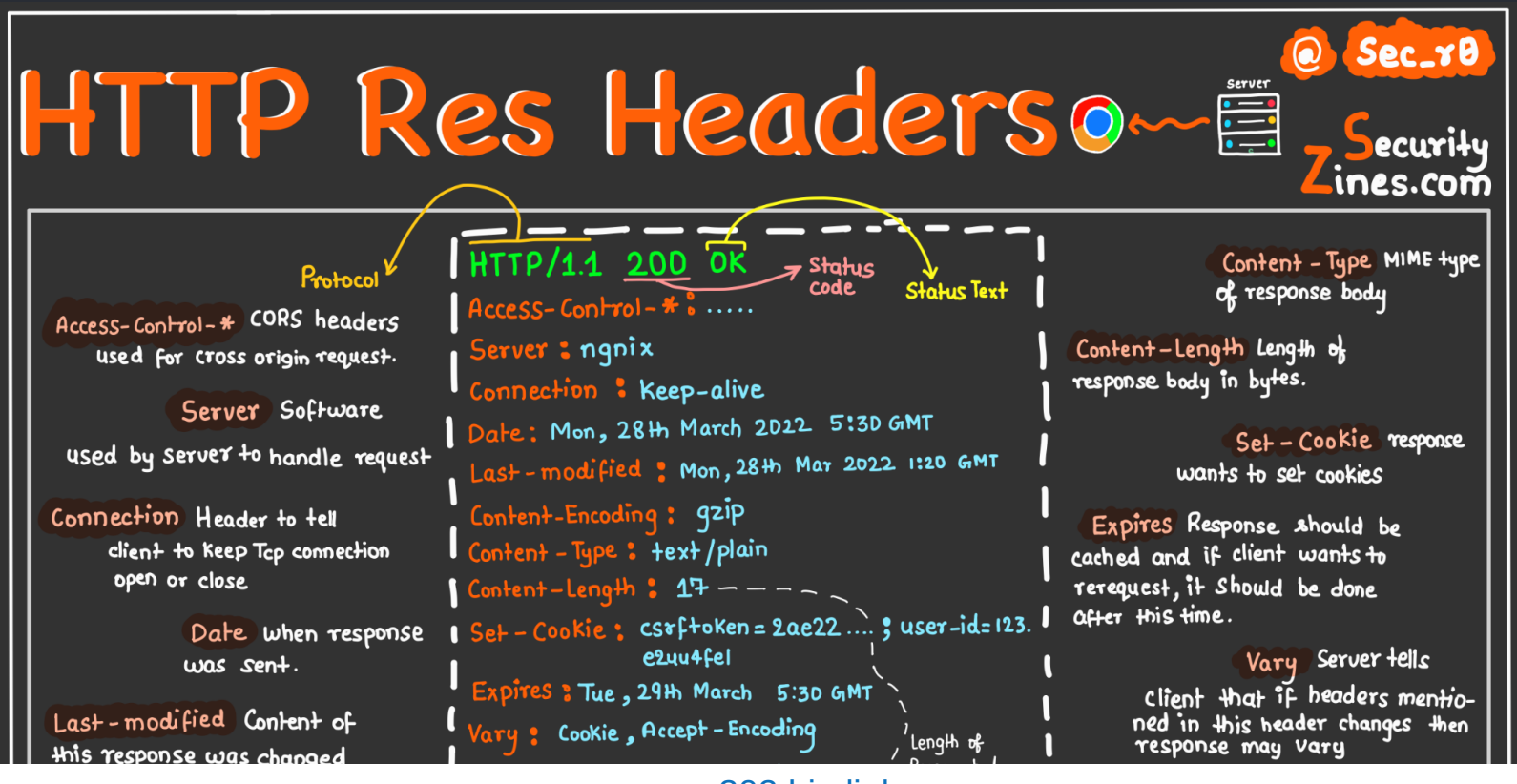
Source

HTTP Request Methods



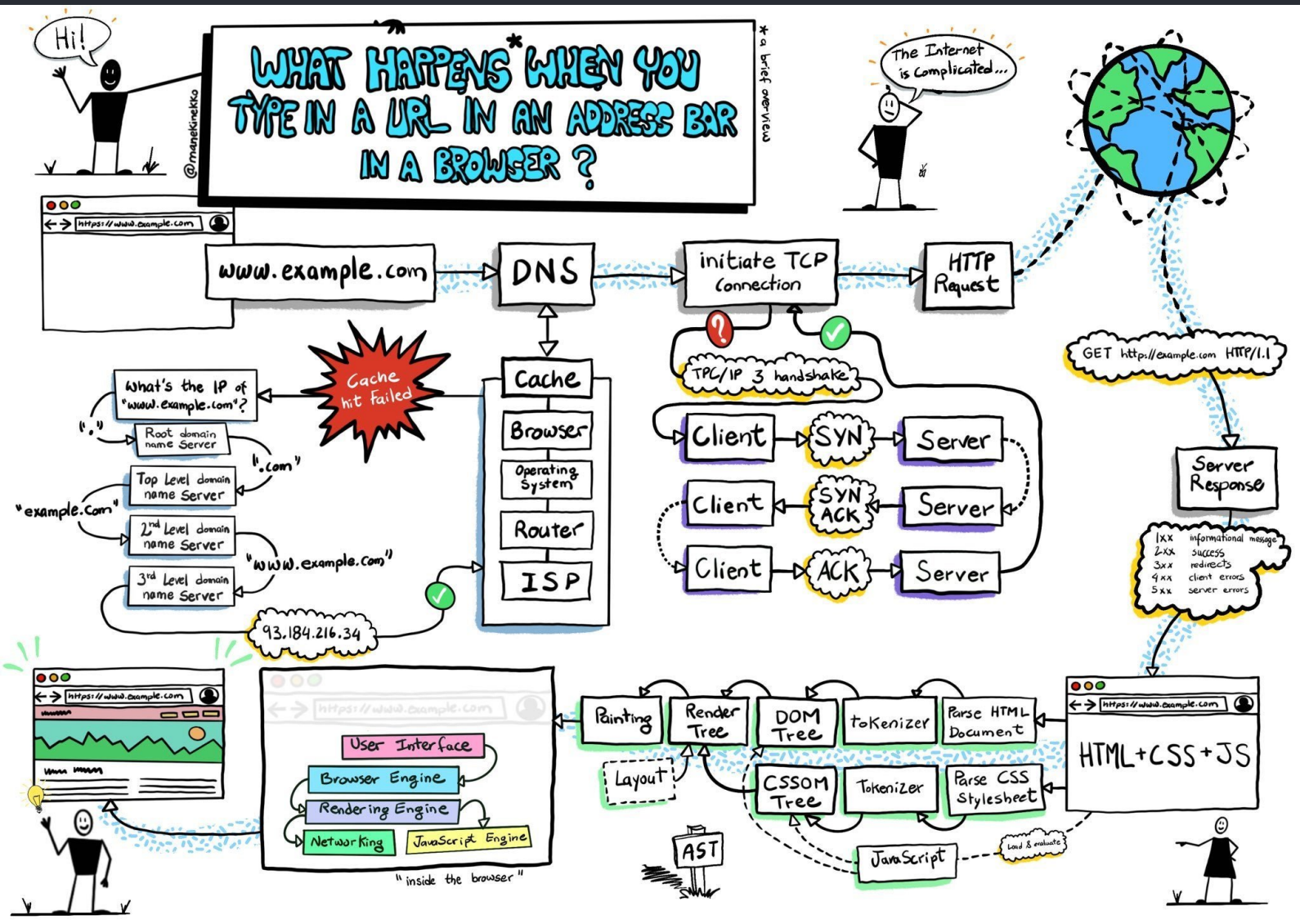
Source

HTTP Response Headers



Source

What Happens when you type a URL in the address bar of a browser ?



Posted in r/**interestingasfuck** by u/**mohiemen**

reddit

Lab Assignment 5 Hints

1. Add at least 2-3 rows of sample data in students record for better debugging.
2. Pass `student_id` parameter for methods doing READ, UPDATE & DELETE operation.
3. `{{loop.index}}` in jinja gives index of each iteration, starting from `1`, helpful for showing courses enrolled.
4. Have fun!, This assignment is cool imo.