1. The main difference between Web 1.0, Web 2.0 and Web 3.0 is primarily modelled on each systems level of user interactivity, but not exclusively. In Web 1.0 resources were made available to users with no opportunity or ability for the user to interact and the user could only read the resource. Initially the resource was also stored on a server.
   The content was static and images, text and the general appearance lacked dynamic ability . There was only one platform for display, a desktop/laptop computer display.

   Web 2.0 became user-centric, where user interactivity becomes the pervasive norm with all resources, making them more application based than static one-way read-only resources. Websites are now fully interactive with major changes added to web1.0 such as :
   U**ser logins**, providing users with their own personal profile one a website which offers major advancements for industries such as banking and personal communication.
   User Generated Content - Resources are no longer exclusively created and generated by the owners thereof, but rather the users themselves are the main generators of the content, for example, YouTube, Facebook, X, TikTok, whereby users are able to create content, communicate on content posted with video, text or audio.
   Improved Infrastructure - The infrastructure of the internet has improved dramatically with servers set up all over the globe, including space to offer a faster and more reliable interconnectivity to humans around the globe with little or no downtime if one server fails.
   Dynamic Content - Resources, thanks to the power of CSS, have become dynamic and more aesthetic to the user. Additionally the advent of the laptop and tablet market, with the proliferation of the mobile phone, now has resources being managed and displayed on multiple devices from one central server, making it convenient and pleasant to access a resource on your desktop, laptop, mobile phone, tablet and even a smart watch or display unit in a vehicle.
   Improved Security - Data are kept on secure servers with security processes such as two-factor authentication, JWS encryption and middleware systems that manage and check user-data on the back-end server side, keeping the browser open for browsing and away from front-end cyber attacks.
   Communication - Thanks to the power of the internet, humans are now able to interact with each globally, in real-time, both in text and audio, (WhatsApp, Messenger etc…) and live, with streaming applications such as Zoom and Teams. The latter was largely due to Covid and the need for humans to interact while in isolation.
   Cloud Based Computing - Resources are now being made for users to store their data on centralised data servers, making it easier for people to access their data from anywhere on earth, versus storing the data on personal servers or back-up drives, such as AWS or Google.

   Web 3.0 is a fluid model, developing in real-time. With all the characteristics of the Web 2.0, the biggest addition to the current web landscape is the inclusion of AI systems that are becoming the norm. User search interactivity on traditional based browser systems like Chrome and Firefox are being surpassed by AI usage. Similarly, no longer are resources just found and presented to users based on crawl index pagination numbers (SERP), but instead, presented with options, explanations, additional links and interactivity. Chrome was not able to rationalise or ‘think’, where LLM models have the ability to learn from previous data encounters and serve up the learnings for future requests. Although in its infancy, AI will become the biggest evolution for the internet and the human race. After the industrial revolution, AI is the next major event on the evolution of the human species. In addition to AI, cloud computing will become the norm, improved infrastructure will further facilitate the speed that we are able to share data and soon, everything will be managed and stored online in some way, shape or form. The banking sector has already become a digital concept.

2. **Front End Development** manages the layer that humans interact with on their devices, i.e., what we experience with our senses. This is generally done with a browser based application that allows users to interact by inputting requests. These requests are then bundled and converted to json and sent to the server where it is parsed back into an object to be used by the server, and then sends it back to the front-end in json again where it is again parsed and turned into a resource that we see or hear. This is done, and managed by providing a framework for the data or resource to be presented on, namely HTML, which builds the

framework of the web display, CSS (Cascading Style Sheets) which helps us manage and control the elements within the HTML environment to allow for dynamism and ease of use and then JavaScript, which allows us to create functions to manage the two-way communication between the back-end and the front-end, deal with user interactivity and additional logic with all the elements involved. Javascript has evolved to work with systems such as Node.js to create powerful web tools such as React, React Mobile and Next.js.

Back-End Development manages the resources that we are trying to access in addition to a host of other functions. Along with many other resources, passwords and user-sensitive information is stored on the back end. All these elements need to be secure and the back-end provides a multitude of tools to manage these such as previously mentioned JWS, Middleware, Cookies, ENV files and more.
The back-end will also serve as the conduit to the database, whether relational (SQL) or non (BSON - MongoDB) and assist in serving up the resources from the database, checking the integrity and matching the request from the HTTP front-end (client) and then responding with the correct resources. Back end systems and frameworks include pHp (Laravel), Node.js(Express) and Python(Django).

3. The HTTP Cycle starts with a user entering a uniform resource locator (URL) into a browser, such as **https://digitalblend.co.za/** where https identifies the protocol being used, the domain name (digital blend), the port is removed (not always displayed) and the base directory of the website. Selecting **https://digitalblend.co.za/weather** would take you to the specific resource for the weather API.
For the HTTP cycle, I'm going to use the example of a local host set server set up on my machine, but the request and response cycle remains the same:

A - User sends a request from the front end to a specific route and end-point on the server using primarily one of 4 C.R.U.D methods namely POST(create), GET(read) PUT(update) & DELETE(delete).

This request is submitted to the server, where it is routed to the correct controller and processed (middleware/user roles assigned/passwords checked etc…) using the request header:

post/http://localhost:8080/api/users/register

The body of the request will contain the information requested in JSON format:

```
{   "email": "test@example.com",
    "password": "your_password"
    "name": “Test”,
    "surname": “User",
    "roleRequest": "admin"
}
```

Once the request has been checked and matches the resource parameters, it will return a HTTP Status, in this case 201 Created. HTTP 100 200 = Initial response, 200 - 300 = success type response, 300 - 400 = redirection response, 400 - 500 = client/server error types and 500 = server errors. These obviously go into more detail for each number i.e. 401, 402, 403 etc…

In the case we have a Server Status of 200 which means the user has been successfully created and a token generated to assign to the JWT in the middleware component matched to this user.

When the user now send the request :

post/http://localhost:8080/api/users/login

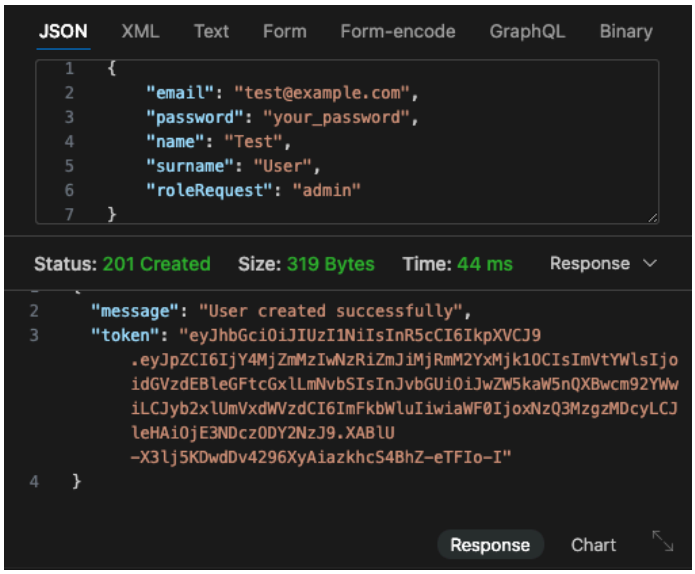
The the request is submitted to the correct router and validated and the response is sent back to the client and the resource is made available to the user.

Register Request

```
let bodyContent = JSON.stringify({ email, password, name, surname, roleRequest });

try {
  let response = await fetch("/api/users/register", {
    method: "POST",
    body: bodyContent,
    headers: headersList
  });
```

Register Response



Login Request

```
let bodyContent = JSON.stringify({ email, passwo

try {
  // Server endpoint
  let response = await fetch("/api/users/login",
    method: "POST",
    body: bodyContent,
    headers: {
```

Login Response