Efolio Week 11 & 12

Student Name: Ziqi Pei

Student Number:33429472

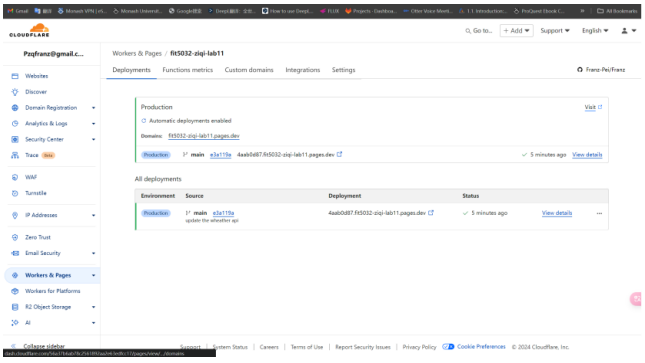
EFOLIO TASK 11.1

Link to your deployed project:

https://fit5032-ziqi-lab11.pages.dev/

Screenshot:

Provide a screenshot of the Cloudflare page indicating it is your deployed project



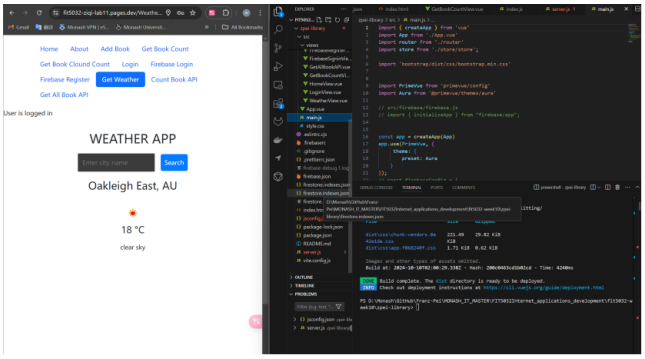
EFOLIO TASK 11.2

Screenshot:

Provide a screenshot of the Cloudflare page, including your address bar, indicating the "Get Weather" functionality is working.

It need modify the weather URL from HTTP to https

```
  iconUrl() {
    return this.weatherData
      ? `https://openweathermap.org/img/w/${this.weatherData.weather[0].icon}.png`
      : null;
  },
},
mounted() {
  this.fetchCurrentLocationWeather();
},
methods: {
  async fetchCurrentLocationWeather() {
    if (navigator.geolocation) {
      navigator.geolocation.getCurrentPosition(async (position) => {
        const { latitude, longitude } = position.coords;
        const url = `https://api.openweathermap.org/data/2.5/weather?lat=${latitude}&lon=${longitude}&appid=${apikey}`;
        await this.fetchWeatherData(url);
      });
    }
```



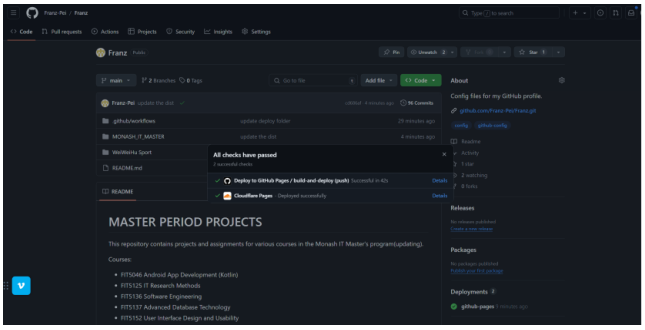
EFOLIO TASK 12.1

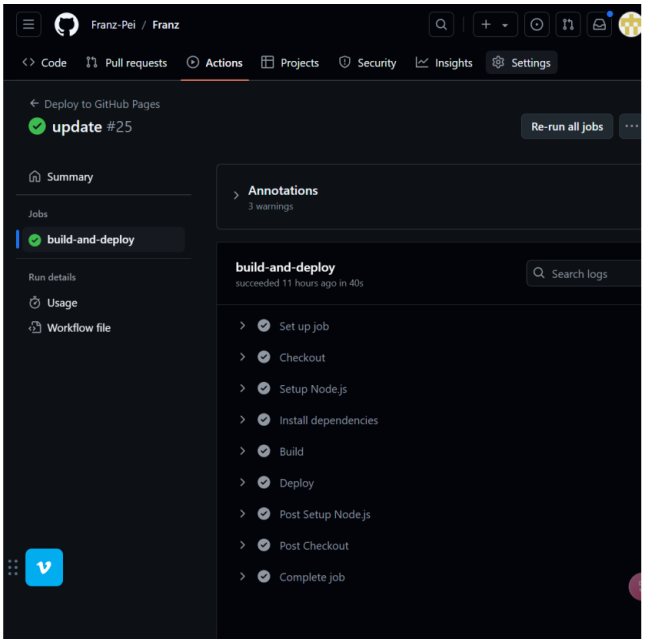
Link to your deployed project:

https://franz-pei.github.io/Franz

Screenshot:

Provide a screenshot of your project's GitHub Actions page indicating that the workflow is running.





EFOLIO TASK 12.2

Text Answer 1: Perform Research on Time to First Byte (TTFB) and How to Reduce TTFB

1. Overview of Time to First Byte (TTFB)

Time to First Byte (TTFB) is a significant metric for assessing website performance, measuring the duration from an HTTP request to the receipt of the first byte from the server. It impacts page speed, user experience, and SEO rankings. Reducing TTFB can effectively enhance the website's overall performance.

1.1 Key Components of TTFB

TTFB is influenced by several stages:

1. **DNS Resolution**

   Resolving domain names into their respective IP addresses.

2. **TCP Connection Establishment**

   Creating a connection between the client and server.

3. **TLS Handshake**

   Establishing a secure connection, particularly with HTTPS.

4. **Server Response Time**

   The time required for the server to process the request and start generating a response.

2. Approaches to Optimize TTFB

2.1 Infrastructure-Level Enhancements

1. **Leverage a Content Delivery Network (CDN)**

   CDNs store website content across multiple geographical locations, reducing latency and expediting the delivery of static assets like images, JavaScript, and CSS.

2. **Upgrade Server Performance**

   - Opt for higher-performance CPUs and faster storage drives.
   - Utilize caching mechanisms to minimize server requests, using HTTP headers and caching query results.
   - Improve database efficiency by reducing slow queries and employing indexing.

3. **Minimize Redirects**

   Reduce unnecessary redirects, especially on essential pages such as the homepage.

4. **Adopt HTTP/2 or HTTP/3 Protocols**

   Use newer HTTP protocols that support multiplexing, which allows multiple requests to be processed over a single connection, decreasing latency.

5. **Implement DNS Prefetching and Preconnect**

   Utilize DNS prefetching and preconnect technologies to resolve DNS names and establish server connections before users click links, minimizing lookup and connection setup delays.

Examples:

```
<link rel="dns-prefetch" href="<https://example.com>">
<link rel="preconnect" href="<https://example.com>" crossor
igin>
```

2.2 Application-Level Improvements

1. **Reduce HTTP Requests**
    - Merge multiple CSS or JavaScript files to reduce request counts.
    - Replace small images with CSS icons to reduce HTTP requests.
    - Enable Gzip or Brotli compression to reduce file sizes, speeding up response times.
2. **Implement Server-Side Rendering (SSR)**

    Generate HTML content on the server and send it to the client, reducing client-side JavaScript execution and improving page load times.

3. Summary

Optimizing TTFB involves leveraging CDNs, improving server performance, reducing redirects, adopting modern HTTP protocols, and implementing DNS prefetching and caching. These strategies help enhance the user experience, page load speed, and SEO outcomes。

Text Answer 2: Perform Research on Hotlinking and How to Prevent It

1. Understanding Hotlinking

Hotlinking refers to unauthorized websites directly linking to resources (images, videos, or files) hosted on your server. This can increase bandwidth usage, degrade server performance, and incur higher operational costs.

1.1 How Hotlinking Occurs

When external websites embed your resources, their users' browsers still make requests to your server to load those assets, placing additional load on your server without authorization.

1.2 Consequences of Hotlinking

1. **Higher Bandwidth Consumption**

    External sites' requests consume bandwidth that you are paying for.

2. **Increased Server Load**

   A large volume of requests can strain the server and negatively affect the user experience of legitimate visitors.

3. **Intellectual Property Infringement**

   Unauthorized use of resources may lead to intellectual property violations, particularly for original content.

2. Techniques to Prevent Hotlinking

2.1 Server-Level Prevention

1. **Adjust Server Configuration to Block Hotlinking**

   Adjust server settings to prevent unauthorized domains from hotlinking. Below is an example using Nginx:

   Example:

```
# Nginx configuration to prevent hotlinking of images
location ~ \\.(jpg|jpeg|png|gif|webp)$ {
    valid_referers none blocked example.com *.example.co
m;
    if ($invalid_referer) {
        return 403;
    }
}
```

2. **Utilize CDN for Hotlink Protection**

   Many CDNs have built-in hotlink protection to block unauthorized domains from accessing your resources.

3. **Add Dynamic Token Authentication**

   Use time-sensitive tokens for resource URLs to prevent other websites from embedding your resources without permission.

4. **Restrict Direct Access to Images**

   Configure the server to allow images to be accessed only via your website, preventing hotlinking through direct URLs.

5. **Load Resources via JavaScript**

Use JavaScript to dynamically load resources, making it harder for external websites to simply copy and use direct links.

6. **Redirect to Custom Warning Images**

   Redirect hotlink attempts to a warning image or information page, letting users know that the resource is being misused.

Example:

```
# Nginx configuration to redirect hotlinking attempts
location ~ \\.(jpg|jpeg|png|gif)$ {
    valid_referers none blocked example.com *.example.com;
    if ($invalid_referer) {
        rewrite ^/.*$ /warning.png redirect;
    }
}
```

3. Summary

Hotlinking can increase server costs, consume bandwidth, and degrade server performance. Preventing hotlinking involves configuring server settings, using CDN protection, adding dynamic token authentication, restricting direct access to resources, and dynamically loading assets. Redirecting unauthorized requests to warning pages can also be an effective deterrent.