SW 오픈소스 활용

-Project Final Reports-

Prepared by

강 형 철

32190103@dankook.ac.kr
(Dept. of Mobile System Engineering)

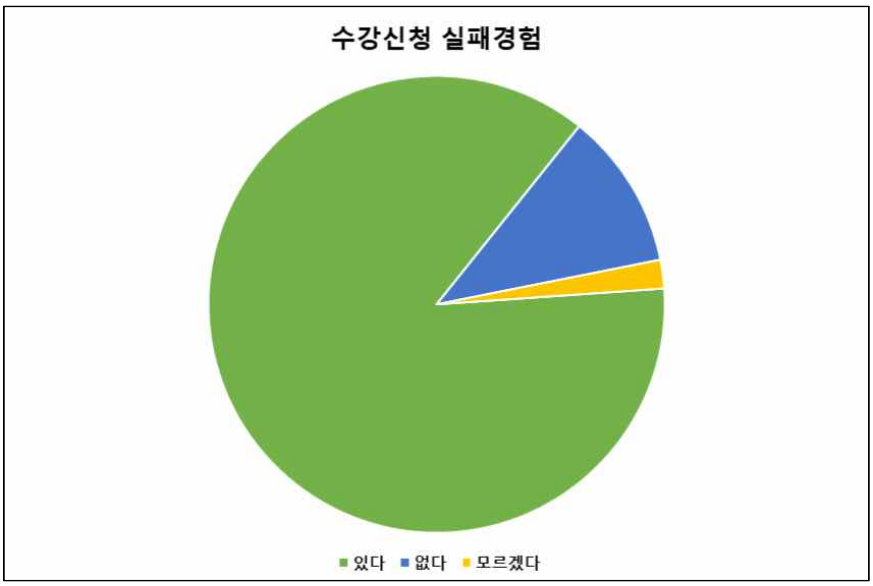
June 07, 2023

목차

1. 프로젝트 배경

이 프로젝트의 기획 배경은, 수강신청과 관련하여 제가 겪었던 일에서 출발했습니다. 2022년 2학기 및 2023학년 1학기, 군 전역 후 복학해 두 학기를 다녔는데, 두 학기 모두 수강신청에 실패하여 전공필수와 같은 필수과목 및 전공선택과목을 포함한 많은 과목의 수강신청에 실패하여 각 과목 교수님에게 강제입력 관련 문의를 많이 했던 것으로 기억합니다. 듣고 싶었던 강의를 인터넷이 느려 서버가 오픈되는 시간을 정확하게 잡지 못했기 때문입니다.

저는 이 경험을 바탕으로, 프로젝트를 진행하기 전에 단국대학교 커뮤니티인 “에브리타임”과 학과 친구들에게 직접 설문을 진행하였습니다. 그 결과 대부분의 학우분들도 저와 비슷한 경험을 한 적이 있다고 응답하였습니다.



<그림1> 수강신청 실패경험 설문결과 - From Proposal of this Projects

따라서, 정확한 서버시간과 인터넷속도 및 핑속도를 측정해주는 사이트를 만들어 내가 직접 사용해 보자! 라는 생각을 가지고 이번 프로젝트를 진행하게 되었습니다. 물론, 비슷한 성격의 사이트인 ‘네이비즘(time.navism.net)’이 이미 존재하지만, 해당 사이트보다 더 정밀한 결과를 출력할 수 있는 사이트를 제작하고자 하였습니다.

이 프로젝트의 소개영상 링크를 첨부해두었으니, 시청하는 것을 추천드립니다.

Project’s Github 주소 : https://github.com/hckang17/ServerTime
Project’s Presentation Movie 주소 : https://youtu.be/eLOD3GHApyo

2. 제공기능

2-1. 주요개념 설명

본 장에서는, 이번 프로젝트 구현 및 설명을 위해 자주 쓰이는 용어에 대해 간략하게 설명하겠습니다.

1) 지연관련

ⓐ 인터넷 지연 : 인터넷 지연은 데이터 패킷이 전송되는 데 걸리는 시간을 나타냅니다. 이는 송신측에서 패킷을 전송한 후 수신측에서 패킷을 받을 때까지의 시간입니다. 일반적으로, 인터넷 지연은 송신 측과 수신 측 간의 라우터, 스위치, 서버 등을 거치면서 발생하는 여러 가지 요인에 의해 발생합니다. 예를 들어, 네트워크 혼잡, 물리적인 거리, 라우팅 문제 등이 인터넷 지연의 원인이 될 수 있습니다. 인터넷 지연은 일반적으로 밀리초 단위로 측정되며, 낮을수록 네트워크 성능이 좋습니다.

ⓑ 패킷 지연(Packet Delay) : 패킷 지연은 네트워크에서 전송되는 개별 패킷의 전달 시간을 나타냅니다. 패킷 지연은 송신 측에서 패킷을 전송한 후 해당 패킷이 수신 측으로 전달될 때까지의 시간입니다. 이는 패킷이 네트워크를 통과하는 동안 발생하는 처리, 전송, 버퍼링 지연 등을 포함합니다. 패킷 지연은 일반적으로 밀리초 단위로 측정되며, 네트워크의 혼잡도, 패킷 크기, 전송 거리 등에 따라 달라집니다.

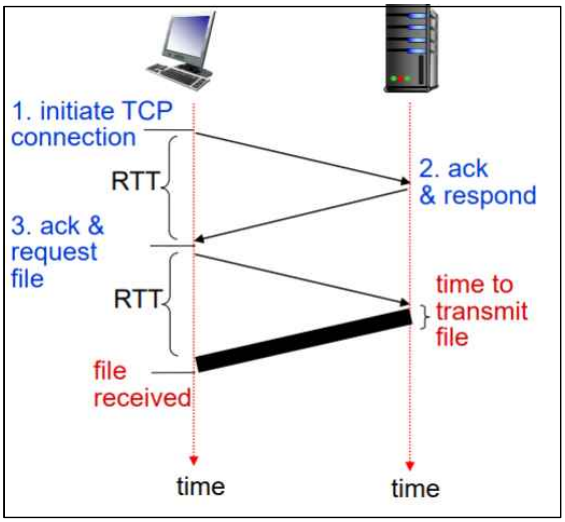
공통점	차이점
- 인터넷 지연과 패킷 지연은 모두 네트워크 통신에서 발생하는 시간 지연 현상을 나타냅니다. - 둘 다 밀리초 단위로 측정되며, 낮을수록 네트워크 성능이 좋습니다.	- 인터넷 지연은 데이터 패킷이 전송되는 데 걸리는 시간을 의미하고, 패킷 지연은 개별 패킷의 전달 시간을 의미합니다. - 인터넷 지연은 송신 측과 수신 측 간의 전체 시간을 나타내는 반면, 패킷 지연은 개별 패킷의 전달 시간만을 나타냅니다. - 인터넷 지연은 네트워크 혼잡, 물리적인 거리, 라우팅 문제 등 다양한 요인에 의해 발생할 수 있습니다. 패킷 지연은 처리, 전송, 버퍼링 지연 등을 포함하여 패킷이 네트워크를 통과하는 동안 발생하는 여러 가지 요인에 의해 발생합니다.

2) 프로토콜 관련

ⓐ HTTP Request/Response : HTTP (Hypertext Transfer Protocol)는 웹 서버와 클라이언트 간의 통신을 위한 프로토콜입니다. HTTP는 클라이언트가 서버에 요청(Request)을 보내고, 서버는 클라이언트에게 응답(Response)을 제공하는 방식으로 동작합니다. 클라이언트는 HTTP Request 메시지를 사용하여 웹 서버에 특정 리소스(예: 웹 페이지, 이미지, 동영상)를 요청하고, 서버는 해당 요청에 대한 적절한 응답을 HTTP Response 메시지로 반환합니다. HTTP Request는 요청 메서드(GET, POST, PUT, DELETE 등)와 헤더, 본문 등의 정보를 포함하며, HTTP Response는 상태 코드(예: 200 OK, 404 Not Found)와 헤더, 본문 등의 정보를 포함합니다. HTTP Request/Response는 ⓓ의 TCP/IP를 이용합니다.

ⓑ ICMP (Internet Control Message Protocol): ICMP는 네트워크에서 메시지를 교환하고 네트워크 상태를 관리하기 위해 사용되는 프로토콜입니다. ICMP는 네트워크 장치들 간의 통신 문제를 진단하고, 에러 메시지를 보내고 받는 데 사용됩니다. ICMP 메시지는 예를 들어, 네트워크 장치의 가용성 확인을 위한 Echo Request와 Echo Reply 메시지(이것을 Ping이라고 부릅니다), 호스트 불응 메시지, 패킷 손실을 보고하는 메시지 등을 포함합니다. ICMP는 IP 프로토콜의 상위 계층에 위치하며, IP 패킷의 데이터 부분에 ICMP 메시지를 캡슐화하여 전송합니다.

ⓒ JSON Request/Response: JSON (JavaScript Object Notation)은 데이터를 구조화하고 전송하기 위한 경량의 데이터 교환 형식입니다. JSON은 텍스트 기반 형식으로, 인간이 읽고 쓰기에 용이하며 기계가 파싱하고 생성하기도 쉽습니다. JSON은 주로 웹 서비스와 클라이언트 간의 데이터 교환에 사용됩니다. JSON Request는 클라이언트가 서버에게 보내는 요청을 JSON 형식으로 표현한 것이고, JSON Response는 서버가 클라이언트에게 반환하는 응답을 JSON 형식으로 표현한 것입니다. 이를 통해 클라이언트와 서버 간의 구조화된 데이터를 교환할 수 있습니다.



<그림2> TCP Handshake Procedure

ⓓ TCP/IP (Transmission Control Protocol/Internet Protocol): TCP/IP는 인터넷을 비롯한 네트워크에서 데이터를 전송하기 위한 프로토콜 스위트입니다. TCP/IP는 인터넷을 포함한 네트워크에서 데이터를 전송하기 위한 프로토콜 스위트입니다. TCP는 신뢰성 있는 연결 지향적인 프로토콜이며, 데이터의 순서와 신뢰성을 보장합니다. IP는 패킷의 라우팅과 전달을 담당하는 프로토콜입니다. TCP/IP에서 데이터 통신은 <그림1>과 같이 Handshake Procedure를 통해 시작됩니다. 클라이언트는 서버에 연결 요청을 보내고, 서버는 연결을 수락합니다. 이를 통해 클라이언트와 서버 간의 연결이 설정됩니다. 이후에 데이터를 요청하는 패킷(여기서는, HTTP Request 혹은 JSON Request)를 서버에 전송하여, 해당 응답을 받습니다.

RTT는 클라이언트와 서버 간 데이터 전송에 걸리는 왕복 시간을 의미합니다. RTT는 네트워크 지연을 측정하고 평가하는 데 사용됩니다. TCP Handshake Procedure에서 RTT는 연결 설정 과정에서 발생하는 지연을 측정합니다.

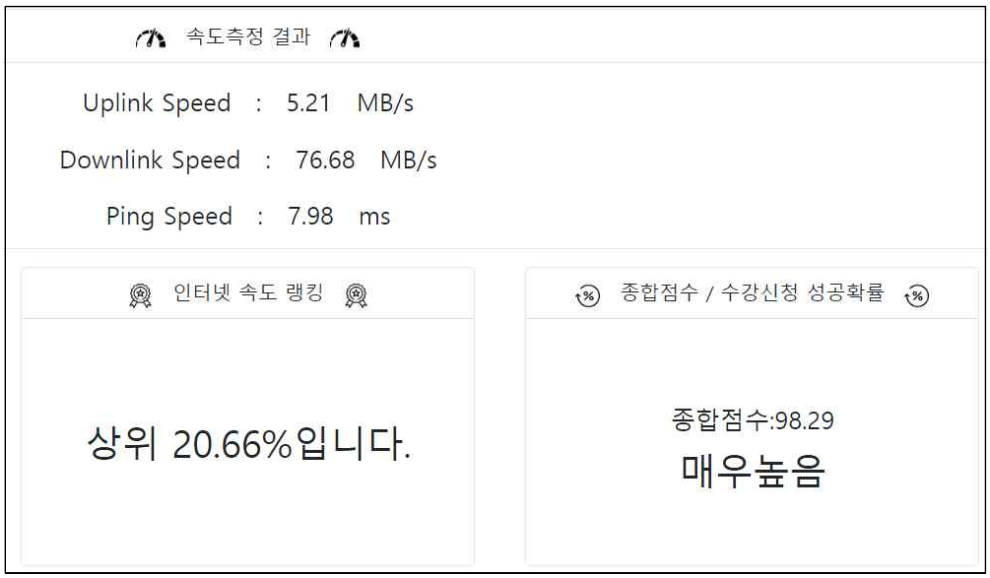
3. 주요 제공기능

3-1. 서버시간

<그림3> 서버시간 표시화면

이 프로젝트에서 제공하는 가장 기본적인 기능입니다. 유저가 확인하고싶은 사이트의 URL을 입력하면, 해당 사이트로 HTTP Request를 보내고, HTTP Response를 받은 뒤, 해당 HTTP Response의 헤더파일에 있는 'DATE' 속성을 가져와 서버시간을 확인하는 형식입니다. 이 작업을, 1000밀리초(1초)마다 반복하여, 유저로 하여금 해당 서버의 서버시간을 확인할 수 있도록 제공합니다. 또한, [2-1]에서 설명한 딜레이를 계산하여 보조지표로 제공합니다.

3-2. 인터넷 속도 측정 및 성공률 자가진단



<그림4> 보조지표 표시화면

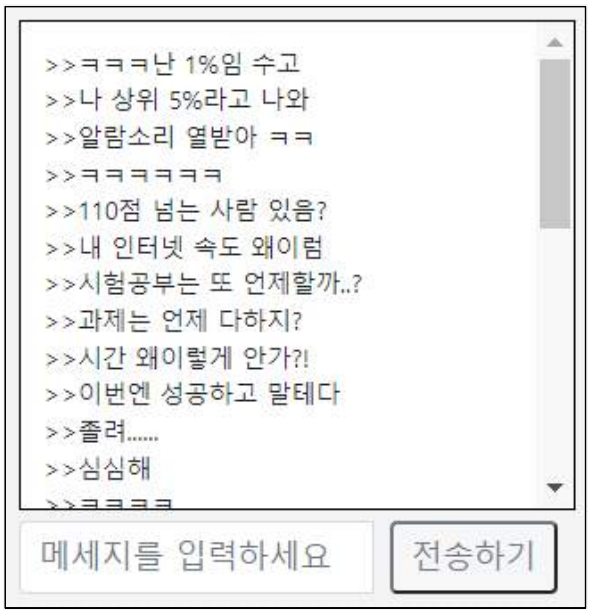
이 프로젝트에서 제공하는 부가적인 기능입니다. 유저가 사용하고 있는 인터넷서비스의 속도관련된 지표(Downlink속도, Uplink속도, Ping속도)를 측정하여, 유저에게 제공합니다. 여기서, 수강신청이나 티켓팅에 직접적으로 영향을 끼치는 지표인 'DownLink속도'와 'Ping속도'정보와 인터넷 속도 랭킹지표를 참고하여 인터넷 상태 종합점수를 평가하고, 그를 바탕으로 **수강신청 확률을 제공**합니다.

3-3. 알람 기능

<그림5> 알람설정 화면

유저가 설정한 시간을 바탕으로 [3-1]의 <그림3>의 시간에 맞춰 알람을 울리는 기능을 제공합니다. 아래의 “3분 전”, “1분 전”, “30초 전”은, 알람설정기준 시간으로부터 해당 시간전에 알람을 울리도록 하여, 혹여 그럴 일은 없겠지만 시간을 놓치는 일이 없도록 제공합니다.

3-4. 채팅 기능

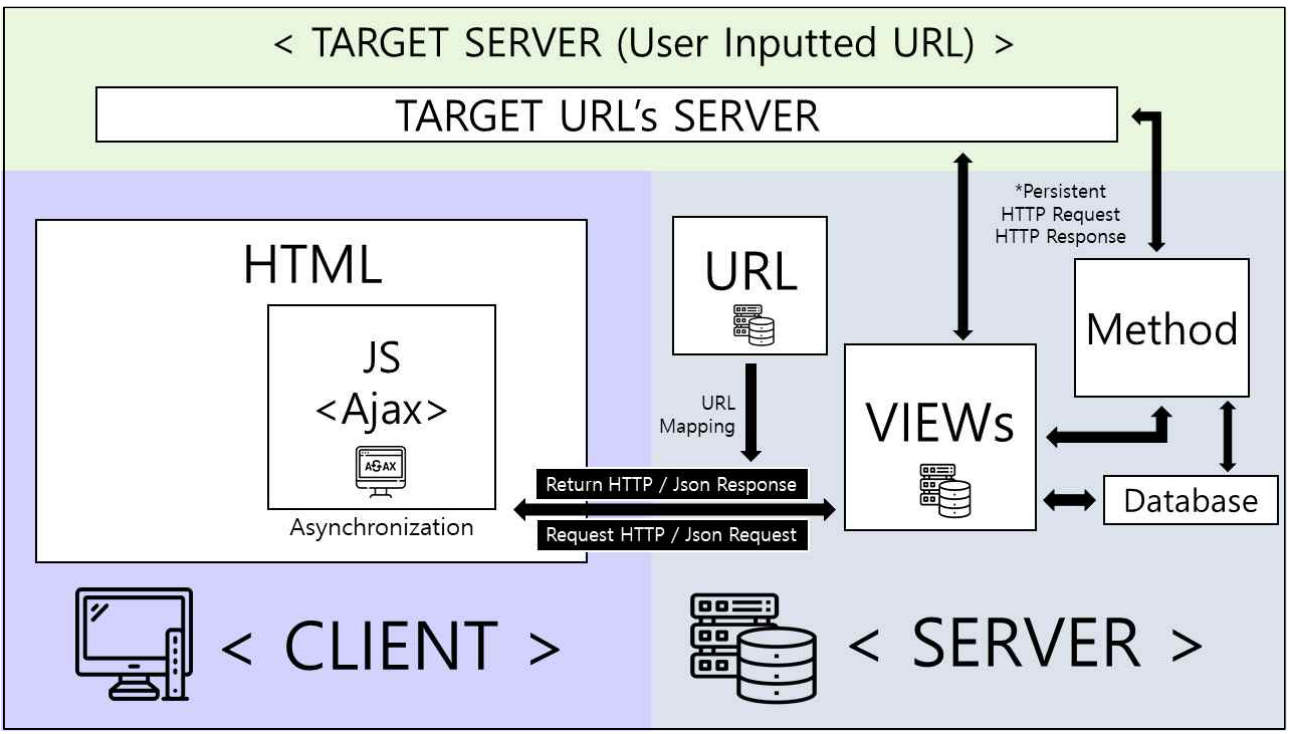


<그림6> 채팅 화면

왼쪽 사이드 메뉴바에서, 대기시간동안에도 본 웹페이지를 이용할 수 있도록 익명으로 실시간 채팅을 할 수 있도록 하였습니다.

4. 구현

4-1. 시스템 구조 및 설계



시스템 구조는 위 그림과 같습니다. CLIENT가 웹사이트에 접속하게 되면, 유저 <SERVER>측으로 필요한 정보를 요청합니다. 이때, 요청하는 정보는 HTTP Request형식으로 매핑되는URL으로 보내지며, View함수는 적절한 정보를 Target Url Server로 다시 HTTP Request를 보내 Response를 받습니다. 이후, 적절한 가공을 통해 CLIENT에게 Json Response혹은 HTTP Response를 돌려주게 됩니다. 이때, 그림의 <METHOD>는 직접적인 연산, 문자열 포맷팅을 맡고 있습니다.

시나리오1) 사용자가 URL을 입력하지 않았을 때

유저 컴퓨터 OS의 시계정보를 가져오는 것이 아니라, 대표적인 NTP서버인 (time.windows.com)에서 시간정보를 가져옵니다. 이때, USER 컴퓨터에서 인터넷 속도 측정이 진행되어, 값을 표시해 줍니다. 필요시, DataBase에 접근하여 정보의 저장 및 불러오기 과정도 진행됩니다. 또한, 채팅의 경우 타겟서버나 디폴트서버에 접근하지 않고, 이 웹페이지를 제공하는 서버와 그 DB에서 데이터 교환이 이루어집니다.

시나리오2) 사용자가 URL을 입력하였을 때

[시나리오1]과 유사하지만, 시간정보를 NTP서버(time.windows.com)에서 가져오는 것이 아니라, Target URL Server측으로 HTTP Request / Response를 받습니다. 인터넷속도 정보는 [시나리오1]과 동일하며, 필요에 따라 DB에 접근하여 적절한 연산을 진행합니다.

4-2. 상세 구현 내용

본 장에서는, 실제로 [3-1~3] 파트의 기능과 [4-1]의 기능을 어떻게 구현했는지 하나씩 살펴보겠습니다.

1) URL MAPPING

```python
from django.contrib import admin
from django.urls import path, include
from . importviews
app_name='config'
urlpatterns= [
    path('main/', include('sugang.urls')),
    path('', views.welcome_page, name='welcome_page'),
]
```

< Config/urls.py >

```python
app_name='sugang'

urlpatterns= [
    path('', views.index, name='main'),
    path('admin/', admin.site.urls, name='admin'),
    path('result/admin/', admin.site.urls),
    path('load_defaultclock/', views.reload_defaultclock, name='defaultclock'),
    path('result/', views.action, name='testURL'),
    path('result/loadclock/', views.reload_serverclock, name='loadclock'),
    path('result/checkUpLink/', views.TestUpLink),
    path('result/checkDownLink/', views.TestDownLink),
    path('result/checkPing/', views.TestPing),
    path('checkUpLink/', views.TestUpLink),
    path('checkDownLink/', views.TestDownLink),
    path('checkPing/', views.TestPing),
]
```

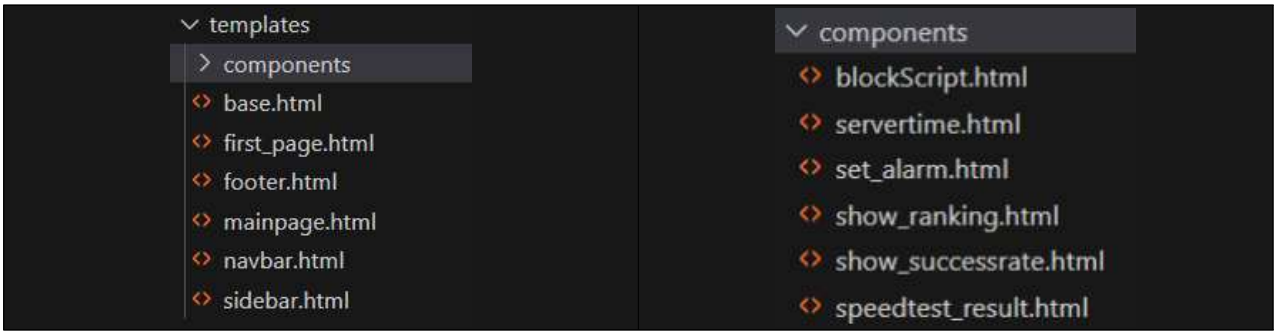
< Sugang/urls.py >

<표2> URL Mapping

<표2>에서와 같이, Django 모체프로젝트인 Config/urls.py에서는 주 기능을 제공하는 main/으로 url을 매핑하였고, main/* 부터는 Sugang 어플리케이션에서 각 기능에 해당하는 View함수를 URL로 매핑하였습니다.

이 중에서, 유저가 직접 사용하게 되는(HTML 매핑이 되어있는 url)은, 'admin/', 'result/'입니다. 나머지는 비동기적으로 화면의 구성요소 값을 갱신해주기 위한 URL입니다.

2) HTML COMPONENTS



페이지수는 많지 않지만, 코드의 길이가 길어 화면 각 구성요소별로 HTML파일을 분리했고, 필요에 따라 적절히 extend, include하여 페이지를 구성하였습니다. 각 html파일별로 어떤 역할을 하는지 설명드리겠습니다.

ⓐ Base.html

```
{% load static %}

<!doctype html>

<htmllang="en">
<head>
  <!-- Required meta tags -->
  <metacharset="utf-8">
  <metaname="viewport"content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <!-- Bootstrap CSS -->
  <linkrel="stylesheet"type="text/css"href="{% static 'style.css' %}">
  <linkrel="stylesheet"type="text/css"href="{% static 'bootstrap.min.css' %}">

  <!-- CSS -->
  <title>SUGANG APPLY</title>
</head>
<body>
  <header>
    <!-- 네비게이션바 -->
   <navclass="navbar navbar-expand-lg navbar-light bg-light border-bottom">
    {% include"navbar.html" %}
   </nav>
    <!-- 네비게이션바 끝 -->
  </header>

  <!-- main -->
  <sectionclass="main-section">
   {% blockcontent %}
   {% endblock %}
  </section>
  <!-- main finish -->

  <!-- 사이드바-->
  <asideclass="main-aside">
   {% include"sidebar.html" %}
  </aside>
  <!-- 사이드바 끝-->

  <!-- 푸터 시작 -->
  <footer>
   {% include"footer.html" %}
  </footer>
  <!-- 푸터 끝 -->

<!-- Bootstrap JS -->
<scriptsrc="{% static 'bootstrap.min.js' %}"></script>


<!-- 자바스크립트 Start -->
{% blockscript %}
{% endblock %}
<!-- 자바스크립트 End -->

</body>
</html>
```

이후, 모든 Html 파일에 공통적으로 포함되어야 하는 태그(<Head><Body>)등을 Base.html에 한번에 선언했습니다. 스타일링을 담당하는 CSS파일 및 JS파일도 여기에 포함되어 있습니다.

ⓑ navbar.html

```
<!-- 모든 페이지 상단에 공통으로 나타날 퀵메뉴바 -->
  <divclass="container-fluid">
   <aclass="navbar-brand"href="{% url'sugang:main'%}">SERVER TIME</a>
   <buttonclass="navbar-toggler"type="button"
     data-bs-toggle="collapse"
     data-bs-target="#navbarSupportedContent"
     aria-controls="navbarSupportedContent"
     aria-expanded="false"
     aria-label="Toggle navigation">
    <spanclass="navbar-toggler-icon"></span>
   </button>
   <divclass="collapse navbar-collapse"id="navbarSupportedContent">
    <ulclass="navbar-nav me-auto mb-2 mb-lg-0">
     <!-- <li class="nav-item"><a class="nav-link" href="">메뉴1</a></li> -->
     <li><aclass="nav-link"href="admin/">관리자모드</a></li>
     <!-- <li><a class="nav-link" href="">메뉴2</a></li> -->
     <li><aclass="nav-link"href="{% url'welcome_page'%}">처음으로</a></li>
    </ul>
   </div>
   <span>
    <ulclass="navbar-nav me-auto mb-2 mb-lg-0">
     <listyle="background-position:right;">
      <aclass="nav-link"href="//github.com/hckang17/ServerTime">개발자 사이트로</a>
     </li>
    </ul>
   </span>
  </div>
```

base.html에 포함되는 navbar.html파일입니다. 모든 웹페이지의 상단에 공통적으로 들어가야할 퀵메뉴를 모아뒀습니다. 스타일링은 부트스트랩을 이용하였습니다. 또한, 이번 프로젝트의 깃헙 주소도 포함시켰습니다.

ⓒ footer.html

```
<div>
  <divstyle="height:10px;"></div>
  <divclass="row justify-content-center ">
   <divstyle="font-size:12px;">
    <p><spanstyle="font-size:15px;">Developed by</span><br>
     32190103 강형철<br>
     Contact Email : <span> hckang17@naver.com </span><br>
       Git  hub  :  <ahref="https://github.com/hckang17/ServerTime"style="text-decoration:
underline; color: black;">github.com/hckang17<br>
     © COPYRIGHTS HCKANG17, EOMLEEHO1 ALL RIGHTS RESERVED</p>
   </div>
  </div>
</div>
```

footer.html 또한, base.html파일에 포함되는 웹페이지 구성요소입니다. <footer>태그 안에 들어갈 내용을 담당하고 있으며, 개발자 정보(제 정보)를 담고있습니다.

ⓓ sidebar.html (1)

```
<divclass="container-fluid">
  <divstyle="height:10px;"></div>
  <divclass="card border-info mb-3"style="max-width: 18rem;">
   <divclass="card-header"style="display: flex; align-items: center; justify-content: center;">
    <div><imgsrc="{% static 'img/timer.png' %}"alt="이미지_설명“
        style="width:20px; height:20px;"></div>
    <divstyle="margin: 0 10px;">알람</div>
    <div><imgsrc="{% static 'img/timer.png' %}"alt="이미지_설명“
        style="width:20px; height:20px;"></div>
   </div>
   <divclass="card-body">
    <h6class="card-title text-center">알람울릴시간</h6>
    <p class="card-text text-center">
      <span id="AlarmNotification"style="font-size:24px;">00:00</span>
      <span id="AlarmNotification-Before"style="font-size:15px;">(미설정)</span></p>
   </div>
  </div>

  <divname="chatbox"class="sidebar-chatbox">
   <spanstyle="font-size:11px; display: block;"id="chattingBox"></span>
  </div>
  <!-- 실시간 채팅 박스 -->
  <divclass="input-group input-group-sm">
   <inputtype="text"class="form-control"placeholder="메세지를 입력하세요“
      id="message_contents">
   <divclass="input-group-append px-2">
    <buttonclass="btn-sm btn-outline-secondary"type="button"id="CommentSendBtn">
      전송하기</button>
   </div>
  </div>

<div>
   <buttonclass="btn btn-sm btn-secondary"type="button“
      data-bs-toggle="offcanvas"data-bs-target="#SpeedTestInformation“
      aria-controls="SpeedTestInformation"style="width:100%;">
    속도측정 정보</button>
  </div>
  <divstyle="height:5px;"></div>
  <div>
   <buttonclass="btn btn-sm btn-secondary"type="button"data-bs-toggle="offcanvas“
      data-bs-target="#SpeedRankingInformation"aria-controls="SpeedRankingInformation“
      style="width:100%;">
    속도랭킹 & 성공확률 정보</button>
  </div>
  <divstyle="height:5px;"></div>
  <div>
   <buttonclass="btn btn-sm btn-secondary"type="button"data-bs-toggle="offcanvas“
      data-bs-target="#ServertimeInformation"aria-controls="ServertimeInformation“
      style="width:100%;">
    서버시간 정보</button>
  </div>

   <!-- 광고 영역 -->
  <divid="ad"class="sidebar-adbox"style="vertical-align:middle;">
   <!-- 광고 내용 -->
   <h6style="vertical-align:middle;">이곳에 광고가 표시됩니다.</h6>
</div>
```

sidebar.html 역시 base.html에 포함되어 웹페이지 공통요소를 구성합니다. <aside>태그내용을 담고있으며, 웹페이지 왼쪽 메뉴바에서 [3-3]의 알람기능과 [3-4]의 채팅기능 및 인터넷속도 측정 및 서버시간 측정과 관련된 정보를 확인할 수 있습니다.

ⓓ sidebar.html (2)

```
<div name="OFFCANVAS INFO">
    <div class="offcanvas offcanvas-start" data-bs-scroll="true" tabindex="-1" id="SpeedTestInformation"
    aria-labelledby="SpeedTestInformationLabel">···
    </div>
    <div class="offcanvas offcanvas-start" data-bs-scroll="true" tabindex="-1" id="SpeedRankingInformation"
    aria-labelledby="SpeedRankingInformationLabel">···
    </div>
    <div class="offcanvas offcanvas-start" data-bs-scroll="true" tabindex="-1" id="ServertimeInformation"
    aria-labelledby="ServertimeInformationLabel">···
    </div>
</div>
```

<그림9> sidebar.html (2)

sidebar.html의 길이가 길어 두 부분으로 나누어 설명하겠습니다.

<그림9>는 사이드바에서 버튼메뉴를 클릭할 시 활성화 되는 “오프캔버스”입니다. <button>요소와 같은 id를 가지는 요소가 활성화 되며, 해당 요소 안의 text요소가 화면에 표시됩니다.

ⓔ servertime.html

```
{%load static%}
<tableclass="table table-light table-striped-colums">
  <thead>
   <tr>
    <thclass="text-center"style="display: flex; align-items: center; justify-content: center;">
     <div><imgsrc="{% static 'img/clock.png' %}"style="width:20px; height:20px;"></div>
     <div>{% ifuser_url %}  '{{ user_url }}'{% else %}  이 사이트{% endif %}의 서버시간  </div>
     <div><imgsrc="{% static 'img/clock.png' %}"style="width:20px; height:20px;"></div>
    </th>
   </tr>
  </tead>

  <tbody>
   <trclass="table-servertime">
    <style>
     .table-servertime {···}
     .table-servertime p{···}
     .table-servertime h3{···}
    </style>
    <tdclass="text-center"style="height:100px; vertical-align:middle;">
     <h2><spanid="servertime-container">{{ current_servertime }}</span></h3>
     <p><spanid="InternetDelay"></span>ms delayed</p>
    </td>
   </tr>
  </tbody>
</table>
```

servertime.html은, mainpage.html에서 서버시간을 표시하는 위젯 부분을 담당하고 있습니다.

view에서 제공받은 {{ current_servertime }}을 표시하고, 이후에는 Ajax로 비동기적으로 갱신됩니다. 아래 Id=“InternetDelay”도 같은 맥락입니다.

ⓕ speedtest_result.html

```
{%load static%}
<divclass="card text-center"style="margin-bottom:15px;">
  <divclass="card-header"style="display: flex; align-items: center; justify-content: center;">
   <div><imgsrc="{% static 'img/speed.png' %}"style="width:24px; height:24px;"></div>
   <div> 속도측정 결과 </div>
   <div><imgsrc="{% static 'img/speed.png' %}"style="width:24px; height:24px;"></div>
  </div>
  <divclass="card-body card-speedresult"style="height:150px;">
   <style>
    .card-speedresult p{
     display: flex;
     justify-content: center;
      font-size:20px;
    }
   </style>
   <pclass="card-text">Uplink Speed : <spanid="uplinkSpeed">측정중입니다...</span> MB/s</p>
   <pclass="card-text">Downlink Speed : <spanid="downlinkSpeed">측정중입니다...</span>
          MB/s</p>
   <pclass="card-text">Ping Speed : <spanid="pingSpeed">측정중...</span> ms</p>
  </div>
</div>
```

speedtest_result.html은 클라이언트의 인터넷 속도 측정 결과를 출력해주는 위젯입니다. AJax로 인해 비동기적으로 요소의 텍스트부분을 갱신해줍니다.

ⓖ show_ranking.html

```
{%load static%}

<style>
  .card-speedrank { … }
  .card-speedrank p { … }
</style>

<divclass="main-rankingbox">
  <divclass="card text-center">
   <divclass="card-header"style="display: flex; align-items: center; justify-content: center;">
    <div><imgsrc="{% static 'img/rank.png' %}"style="width:20px; height:20px;"></div>
    <div> 인터넷 속도 랭킹 </div>
    <div><imgsrc="{% static 'img/rank.png' %}"style="width:20px; height:20px;"></div>
   </div>
   <divclass="card-body card-speedrank"style="height:200px;">
    <divstyle="display: flex; flex-direction: column; justify-content: center;">
     <pclass="card-text"style="margin: 0 auto;"><spanid="speedRanking">측정중...</span></p>
    </div>
   </div>
  </div>
</div>
```

show_ranking.html은 측정된 인터넷속도를 통해 클라이언트의 인터넷이 이용자 상위 몇%인지 계산된 수치를 표시해주는 위젯입니다. 변동사항이 생긴다면, 요소에 비동기적으로 수치가 갱신됩니다.

ⓗ set_alarm.html

```
{%load static%}
<divclass="main-alarmbox">
  <divclass="card">
    <divclass="card-header text-center"style="display: flex; align-items: center; justify-content:
        center;">
    <div><imgsrc="{% static 'img/timer.png' %}"style="width:20px; height:20px;"></div>
    <div> 알람 설정하기 </div>
    <div><imgsrc="{% static 'img/timer.png' %}"style="width:20px; height:20px;"></div>
   </div>
   <divclass="card-body"style="height:200px;">
   <formid="alarm-form">
     <divclass="form-group">
      <inputtype="time"class="form-control"id="alarm-time">
     </div>
     <divclass="form-group">
      <label>알람 울릴시간</label>
     <divclass="form-check">
      <inputtype="radio"class="form-check-input"id="set_3min"name="set_alarm"value="180">
      <labelclass="form-check-label"for="set_3min">3분 전</label>
     </div>
     <divclass="form-check">
      <inputtype="radio"class="form-check-input"id="set_1min"name="set_alarm"value="60">
      <labelclass="form-check-label"for="set_1min">1분 전</label>
     </div>
     <divclass="form-check">
      <inputtype="radio"class="form-check-input"id="set_30sec"name="set_alarm"value="30">
      <labelclass="form-check-label"for="set_30sec">30초 전</label>
     </div>
    </div>
     <buttontype="submit"class="btn btn-secondary"style="width:100%; margin:auto;">알람 설정
        </button>
   </form>
   </div>
  </div>
</div>
```

set_alarm.html은 클라이언트가 알람을 설정하기 위한 <form>요소를 담고있습니다. [알람 설정] 버튼 요소를 클릭하게 되면, method="POST"형식으로, 서버에 기준시간과 '몇 분 전'요소가 전달되고, 서버는 해당 시간정보를 가지고 알람을 작동시킵니다.

ⓘ mainpage.html

```
{% extends'base.html' %}
{% load humanize %}
{% load static %}

{% blockcontent %}
  <!-- 확인하고자 하는 URL 입력창 &버튼 -->
  <formmethod="post"action="{% url'sugang:testURL' %}">
   {% csrf_token %}
   <divclass="input-group mb-3 URL-input">
    <buttonclass="btn btn-outline-secondary"type="submit">테스트</button>
    <inputid="saveURL"name="saveURL"type="text"class="form-control px-3"placeholder="접속하고
자 하는 URL을 입력해주세요.">
   </div>
  </form>

  <sectionclass="result-table">
   {% include"components/servertime.html" %}
   {% include"components/speedtest_result.html" %}
  </section>

  <sectionstyle="display: flex; align-items: center; justify-content: center; height: 260px;">
   {% include'components/set_alarm.html' %}
   {% include'components/show_ranking.html' %}
   {% include'components/show_successrate.html' %}
  </section>
{% endblock%}

{% blockscript %}
  {% include'components/blockScript.html'%}
{% endblock %}
```

mainpage.html은 base.html을 extend(상속)하여 사용함으로, base.html의 모든 요소를 포함하고 있습니다. mainpage.html은 base.html에서 {% block content %} 부분과 {% block script %}를 재정의해서 사용합니다. 또한, ⓔ~ⓗ의 모든 위젯을 포함하고 있습니다.

3) VIEW & METHODS

본 장에서는 각 URL에 매칭되어있는 View함수와, 내장 메서드들을 살펴보겠습니다.

렌더링에 직접적으로 관여하는 함수들은 views.py에, 그 외에 단순 계산이나 DB접근에 관여하는 함수들은 function/method.py에 구현하였습니다.

3-1) Views.py

```
1   import datetime
2   import time
3   import ntplib
4   import ping3
5   from django.http import HttpResponse
6   from django.shortcuts import get_object_or_404, redirect, render
7   from .models import *
8   from .functions import method
9   from django.http import JsonResponse
10
11  # Create your views here.
12
13  def index(request):…
18
19  def action(request):…
35
36  def reload_serverclock(request):…
47
48  def reload_defaultclock(request):…
61
62  def TestUpLink(request):…
66
67  def TestDownLink(request):…
88
89  def TestPing(request):…
94
95  def send_message(request):…
105
106 def read_message(request):…
```

View.py는 총 9개의 함수로 구성되어 있습니다. 이 중, 두 가지만 Render와 관련이 있고 나머지는 자바스크립트와 연관되어, 비동기적으로 HTML구서용소를 갱신하기 위한 JsonResponse를 반환하는 함수들입니다.

ⓐ def index(request):

```
def index(request):
    current_time=datetime.datetime.now()
    formatted_time=current_time.strftime("%Y년 %m월 %d일 %H시 %M분 %S초")
    context= {'current_servertime' : formatted_time}
    return render(request, 'mainpage.html', context)
```

index함수는, ‘main/’의 URL에 대응하는 View함수로, User가 아무 URL도 입력하지 않았을 표시될 화면을 구성하는 함수입니다. Django문법으로 Dictionary형 context를 매개변수로 넘겨서 HTML이 해당 요소를 추출하여 사용할 수 있도록 하였습니다.

ⓑ def action(request):

```
def action(request):
    context= {}
    if request.method =='POST':
      if request.POST.get('saveURL') =='':
        return redirect('sugang:main')
      else:
        method.save_URL(request)
        temp=method.show_server_time(method.get_accessurl_by_highest_id())
        context['current_servertime'], context['user_url'] =temp[0], temp[1]
        return render(request, 'mainpage.html', context)
    else:
      return redirect('sugang:main')
```

action함수는 사용자가 입력창에 URL을 입력하고, [테스트] 버튼을 클릭하였을 때 작동하는 함수입니다. 차후의 데이터활용을 위해 우선, 유저인풋을 DB에 저장합니다. 이후, 저장된 데이터를 활용하여 서버시간을 계산합니다. 이때, method.show_server_time()함수가 호출됩니다.

눈여겨 볼 것은, 여기서는 사용자가 버튼을 클릭했을 때의 서버시간을 반환한다는 것입니다. [4-3]에서 설명하겠지만, Django로는 비동기적으로 컴포넌트를 갱신할 수 없기 때문에 자바스크립트를 통해 서버시간을 출력합니다.

ⓒ def reload_serverclock(request):

```
def reload_serverclock(request):
    start_time=time.time()
    target_url=request.POST.get('targetURL')
    print(target_url)
    server_time=method.calculate_time(target_url)
    end_time=time.time()
    run_time=end_time-start_time
    run_time=run_time/4  # 2RTT가 결국 InternetDelay이기 때문에 실제 서버시간 오차는 0.5RTT임.
    run_time=round(run_time*1000, 2)
    print("Load Clock time : ", str(run_time))
    return JsonResponse({'current_servertime':server_time, 'InternetDelay':run_time})
```

reload_serverclock()함수는, 실시간으로 서버시간 가져오기 위한 함수로, 비동기적 구성요소 갱신을 위한 JsonResponse를 반환합니다. 이 과정에서, HTTP Request가 발생하게 되는데, [2-1]에서 언급했듯 HTTP Request/Response는 TCP/IP를 사용합니다. 따라서, 2RTT를 계산하여 실제 서버시간과의 오차가 몇 밀리초인지 계산하여 서버시간과 함께 반환합니다.

※ JsonResponse는 Dictionary타입으로 반환받아야, 이후 JS에서 쉽게 파싱해서 사용할 수 있습니다.

ⓓ def reload_defaultclock(request):

```
def reload_defaultclock(request):
    start_time=time.time()
    ntp_server='time.windows.com'
    client=ntplib.NTPClient()
    response=client.request(ntp_server)
    ntp_time=datetime.datetime.fromtimestamp(response.tx_time)
    formatted_time=ntp_time.strftime("%Y년 %m월 %d일 %H시 %M분 %S초")
    end_time=time.time()
    run_time=end_time-start_time
    run_time=run_time/4  # 2RTT가 결국 InternetDelay이기 때문에 실제 서버시간 오차는 1RTT임.
    run_time=round(run_time*1000, 2)
    print("Load DefaultClock time : ", str(run_time))
    return JsonResponse({'current_servertime':formatted_time,'InternetDelay':run_time})
```

reload_defaultclock함수는, OS의 시간을 참조합니다. 이때, 사용자 컴퓨터의 OS 시간을 가져오는 것이 아니라, NTP서버에서 표준시간을 가져오게 됩니다. 이를 위해서, ntplib 라이브러리를 활용하였으며, NTP서버로는 'time.windows.com'을 이용하였습니다.

시간측정 매커니즘은 ⓒ에서와 동일합니다.

ⓔ TestUpLink / TestDownLink / TestPing

```
def TestUpLink(request):
    upSpeed=method.checkUpLink()
    response= {'upSpeed':upSpeed}
    return JsonResponse(response)

def TestDownLink(request):
    pingSpeed=request.POST.get('pingSpeed')
    try:
      pingSpeed=float(pingSpeed)   # 시도하여 float으로 변환
    exceptValueError:
      pingSpeed=30
    print(pingSpeed)
    downSpeed=method.checkDownLink()  # 다운로드속도 확인
    result=resultInfo.objects.create(
      upSpeed=0,
      downSpeed=downSpeed,
      pingSpeed=0)
    down_percentile=method.get_speed_percentile(downSpeed)   # 다운로드속도 순위 측정
    probability_success, score=method.get_success(down_percentile, pingSpeed, downSpeed)
    down_percentile_str="상위 {:.2f}%입니다.".format(down_percentile)
    score_str="종합점수:{:.2f}".format(score)
    result.save()
    response= {'downSpeed':downSpeed, 'speed_ranking':down_percentile_str,
               'success':probability_success, 'score':score_str}
    return JsonResponse(response)

def TestPing(request):
    hostname='time.bora.net'
    response_time=round(ping3.ping(hostname) *1000, 2)
    response= {'pingSpeed':response_time}
    return JsonResponse(response)
```

TestUplink / TestDownlink / TestPing은 각각, 업로드속도, 다운로드 속도, 핑속도를 method.checkUpLink, method.checkDownLink 메서드로 측정하는 함수입니다.

모두 JsonResponse를 반환하며, TestDownlink함수만, DB에 ResultInfo객체를 저장합니다. 또한, method.get_speed_percentile()함수를 호출하여, 인터넷 속도 랭킹을 계산합니다. 이후, method.get_success함수를 호출하여 인터넷상태점검 종합점수 및 수강신청 성공확률을 보여줍니다. 실질적인 측정 및 계산 매커니즘은 (3-2) Method.py 파트에서 설명하겠습니다.

ⓕ send_message / read_message

```
def send_message(request):
    ifrequest.method =='POST':
        message=request.POST.get('message_content')
        # Comment 모델을 사용하여 댓글 저장
        print("입력"+message)
        comment=Comment(content=message)
        comment.save()
        return JsonResponse({'status': 'success'})
    else:
        return JsonResponse({'status': 'error'})

def read_message(request):
    Comment_lists=Comment.objects.order_by('-created_at')[:30]
    Comment_lists_str=''
    for data in Comment_lists:
        Comment_lists_str+=">>{}<br>".format(data.content)
    response= {'Comment_list':Comment_lists_str}
    return JsonResponse(response)
```

위 두 함수는, 채팅기능에 사용되는 함수입니다. 물론 두 함수는 자바스크립트를 호출하여

send_message부터 간편히 살펴보면, 유저가 버튼을 누르면 Ajax로 send_message함수가 매핑되어있는 URL에 요청(Request)를 보내고, send_message가 요청을 받아, Comment객체를 DB에 저장하고 JsonResponse를 리턴하는 것을 볼 수 있습니다.

read_message도 위와 비슷한 매커니즘이지만 DB로부터 Comment를 'created_at' 필드를 기준으로 내림차순 정렬하여, 상위30항목만 불러와 ">>{}
".format(data.content)형식으로 바꿔 JsonResponse로 보내는 것을 확인할 수 있습니다.

3-2) Method.py (1)

주로 View함수에서 반환해야하는 값을 직접 계산하는 함수들을 모아놓은 파일입니다.

```
from ..models import accessURL
import urllib.request
import urllib.error
import datetime
from .. views import*
import speedtest_cli
import numpy as np
import math

def get_accessurl_by_highest_id():
    highest_id=accessURL.objects.order_by('-id').values_list('id', flat=True).first()
    if highest_id is not None:
        highest_id_accessurl=accessURL.objects.get(id=highest_id)
        return highest_id_accessurl
    else:
        return None

def calculate_time(targetURL):  #targetURL 은 URLtype
    response=urllib.request.urlopen(targetURL).headers['Date']
    datetime_obj=datetime.datetime.strptime(response, "%a, %d%b %Y %H:%M:%S %Z")
    datetime_obj+=datetime.timedelta(hours=9)
    # 원하는 형식으로 포맷팅
    formatted_date=datetime_obj.strftime("%Y년 %m월 %d일 %H시 %M분 %S초")
    return formatted_date

def save_URL(request):
    testURL=request.POST.get('saveURL')
    if not testURL.startswith('https://'):
        testURL='https://'+testURL
    saveURL=accessURL(testURL=testURL)
    saveURL.save()

def show_server_time(saveURL):
    try:
        targetURL=saveURL.testURL
        serverTime=calculate_time(targetURL)
        print(serverTime, targetURL)
        return serverTime, targetURL
    except:
        return HttpResponse("Could not retrieve server time.")

def checkSpeed():
    st=speedtest_cli.Speedtest()
    st.get_best_server()
    up_speed=round(st.upload() /1000000 , 2)
    down_speed=round(st.download() /1000000 , 2)
    ping_speed=st.results.ping
    return up_speed, down_speed, ping_speed
```

ⓐ get_accessurl_by_id() 함수는, DB에 있는 accessURL객체중에 가장 최근에 저장된 객체를 찾아, 그 객체를 반환하는 함수입니다.

ⓑ calculate_time() 함수는, targetURL로 HTTP 요청을 보내고, 응답으로부터 'Date' 필드의 값을 추출하여 날짜와 시간 정보를 가져오는 과정을 수행합니다. 추출한 날짜와 시간 정보는 datetime 객체로 변환되고, 이후에 9시간을 더하여 시간을 조정하여, 적절한 형식으로 포맷팅하여 반화합니다.

ⓒ save_URL함수는, 유저로부터 입력받은 URL을 accessURL객체로 담아 DB에 저장합니다.

ⓓ show_server_time() 함수는, saveURL을 인자로 받아 서버 시간을 가져옵니다. 함수 내부에서는 targetURL 변수에 saveURL 객체의 testURL 속성 값을 할당합니다. 그리고 calculate_time 함수를 호출하여 targetURL을 인자로 전달하여 **서버 시간을 가져옵니다**. 가져온 서버 시간과 targetURL을 출력하고, 해당 값들을 반환합니다. 만약 예외가 발생하여 서버 시간을 가져오지 못하면 "Could not retrieve server time." 메시지를 포함한 HttpResponse 객체를 반환합니다.

ⓔ checkSpeed() 함수는, 인터넷 속도를 측정합니다. 함수 내부에서는 speedtest_cli 모듈을 사용하여 속도 측정 객체 st를 생성합니다. st.get_best_server()를 호출하여 최적의 서버를 선택합니다. 그 후, st.upload()를 호출하여 업로드 속도를 측정하고, 그 값을 밀리바이트 단위로 변환하여 up_speed 변수에 할당합니다. 마찬가지로, st.download()를 호출하여 **다운로드 속도를 측정**하고, 그 값을 밀리바이트 단위로 변환하여 down_speed 변수에 할당합니다. 마지막으로, st.results.ping을 통해 핑(Ping) 속도를 가져와 ping_speed 변수에 할당합니다. 최종적으로 up_speed, down_speed, ping_speed 값을 반환합니다.

※ 하지만, speedtest_cli 라이브러리를 통해 한번에 세가지 정보를 다 가져올 경우, **실행시간이 30초정도 소요**되기 때문에, 최적화를 위해 다시 한번 분리하여 사용하였습니다. 이 다음파트에서 설명하겠습니다.

3-2) Method.py (2)

(1)에 이어서, 설명하겠습니다.

```
def get_speed_percentile(down_speed):
  # 데이터베이스에 저장된 다운로드 속도 데이터를 가져옵니다.
  speed_data=resultInfo.objects.all().values_list('downSpeed')
  speed_data= [item[0] for item in speed_data]
  # 속도 데이터가 존재하지 않으면 0을 반환합니다.
  if not speed_data:
   return 0
  # 속도 데이터가 존재할 경우 각 속도에 대한 순위를 계산하고, down_speed의 순위를 구합니다.
  rank=sum(down_speed>s for s in speed_data)
  percentile=round(rank/len(speed_data) *100, 2)
  percentile=round(100-percentile, 2)
  return percentile

def checkUpLink():
  st=speedtest_cli.Speedtest()
  st.get_best_server()
  up_speed=round(st.upload() /1000000 , 2)
  return up_speed

def checkDownLink():
  st=speedtest_cli.Speedtest()
  st.get_best_server()
  down_speed=round(st.download() /1000000 , 2)
  return down_speed

def checkPing():
  st=speedtest_cli.Speedtest()
  st.get_best_server()
  return round(st.results.ping, 1)
```

ⓕ get_speed_percentile() 함수는 다운로드 속도의 백분위수(percentile)를 계산합니다. 함수 내부에서는 데이터베이스에 저장된 다운로드 속도 데이터를 가져와 speed_data 변수에 할당합니다.

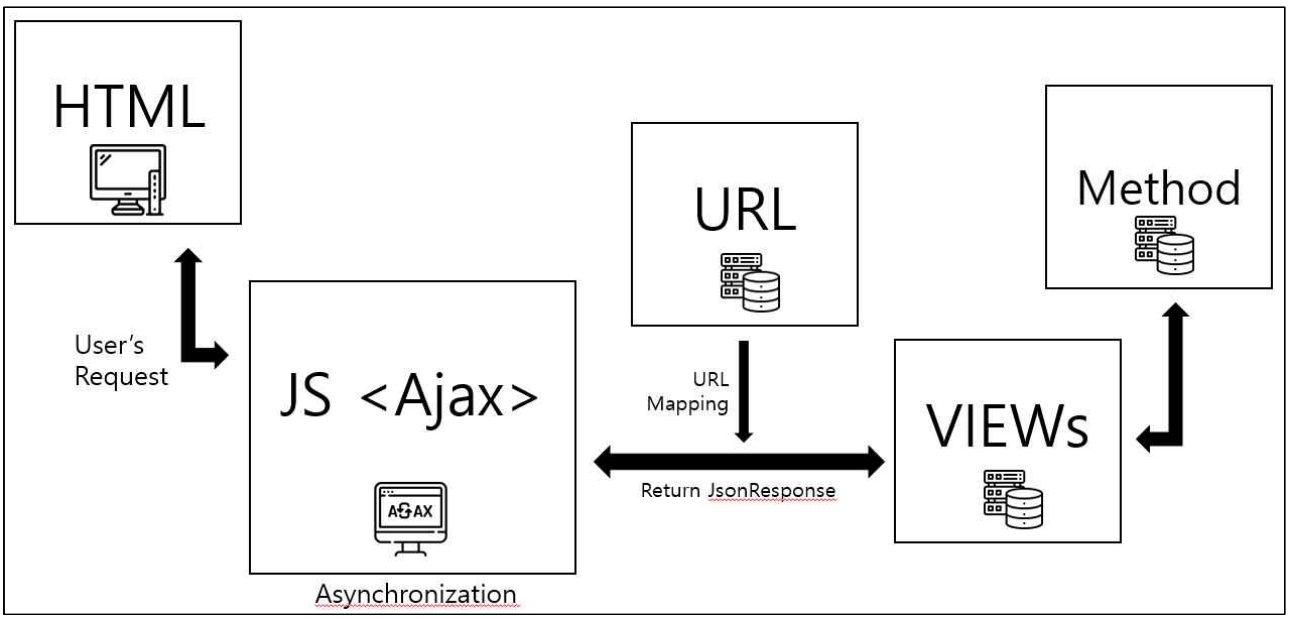
이후, speed_data에서 각 속도 값을 가져와 리스트로 변환합니다. 속도 데이터가 존재하지 않을 경우 0을 반환합니다. **속도 데이터가 존재하는 경우, down_speed보다 작은 속도 값의 개수를 계산하여 rank 변수에 할당합니다.** 그리고 rank를 전체 속도 데이터 개수로 나누어 백분위수를 계산합니다. 최종적으로 percentile 변수에 할당하고, 100에서 percentile을 뺀 값을 다시 percentile 변수에 할당합니다. 마지막으로, 인터넷속도 백분위값인 percentile 값을 반환합니다.

※ 인터넷 속도값 데이터는, **이 사이트를 이용하여 인터넷 속도를 측정한 모든 유저들의 기록**을 모두 모아서 만든 자체 데이터베이스입니다.

ⓖ checkUPLink()함수와 checkDownLink()함수 및 checkPing()함수는, ⓔ의 checkSpeed()함수를 각 측정단위별로 분리하여 만든 함수입니다. 이를 통해, 실행시간(측정시간)을 대폭 감소시킬 수 있었고, Ajax를 활용하여 비동기적으로 병렬처리도 가능해 더욱 효용성 높게 사용할 수 있었습니다.

4) JavaScript

[4-2]의 HTML Component를 확인해보면, “blockscript.html”파일이 존재한다는 것을 확인할 수 있습니다. 이 웹페이지를 구성하고있는 모든 <script>파일이 이 파일안에 정의되어 있으며, 사실상 이번 웹 사이트의 핵심이라고 할 수 있습니다. JavaScript와 View, Method가 맞물려서 HTML의 각 요소를 비동기적으로 갱신하는 매커니즘은 아래와 같습니다.



HTML의 버튼 등의 트리거로, JS가 실행되는데 이때 Ajax를 통해 HTTP Request를 보낼URL을 결정합니다. 이후, 해당 URL로 HTTP Request가 전달되는데 이때 Django의 MVC모델이 적용됩니다. urls.py에서 View함수를 매칭해준 뒤, VIew함수에서 Method.py와 함께 필요한 결과를 도출하여, JS에게 JsonResponse를 돌려주면, <Ajax>요소가 특정 HTML 구성요소의 값(주로 id로 탐색)을 비동기적으로 갱신합니다.

ⓐ function updateClock()

```
function updateClock() {
   var loadClockURL='{% if user_url %}loadclock/{% else %}load_defaultclock/{% endif %}';
   var targetURL='{{ user_url }}';

   $.ajax({
    url:loadClockURL,
    method:'POST',
    headers: { "X-CSRFToken": csrftoken, "Connection":"keep-alive" },
    data: { 'targetURL':targetURL }, // 키 이름이 'targets'가 아닌 'targetURL'로 수정되었습니다.
    success:function(data) {
     $('#servertime-container').text(data.current_servertime);
     $('#InternetDelay').text(data.InternetDelay);
     current_time_ajax=data.current_servertime;
    },
    error:function(xhr) {
     console.log(xhr.status+': '+xhr.statusText);
    }
   });
  }
```

updateClock()함수는, 위 표에서는 보이지 않지만 setInterval(updateClock, 1000)형태로 1000밀리초에 한번 실행 되도록 되어있습니다.

```
var loadClockURL='{% if user_url %}loadclock/{% else %}load_defaultclock/{% endif %}';
```

Django 문법을 활용하여, {{ user_url }} 이 있을 때 없을때의 loadClockURL값을 달리 저장합니다. 이후, var targetURL='{{ user_url }}'; 부분에서, 유저가 입력한 URL을 가져옵니다.

그 다음, $.ajax를 통해, loadClockURL에 HTTP Request를 발송합니다.

```
headers: { "X-CSRFToken": csrftoken, "Connection":"keep-alive" },
```

이때, csrf토큰을 활용하여 보안성을 높입니다. 또한, HTTP Connection을 Keep-Alive으로 설정하여, 반복적인 HTTP Request에서 Persistent한 HTTP Request로 하여 매번 새로운 TCP 커넥션(Hand SHaking)과정을 생략해 실행시간을 줄이도록 하였습니다.

이후로 HTTP Request에 'targetURL'을 넣어줌으로써, View함수가 Request.POST.get('targetURL')로, 해당 정보를 사용할 수 있도록 하였습니다.

```
success:function(data) {
  $('#servertime-container').text(data.current_servertime);
  $('#InternetDelay').text(data.InternetDelay);
  current_time_ajax=data.current_servertime;
 }
```

마지막으로, 성공적으로 JsonResponse가 도착했다면, #으로 표시된 id요소의 값을 갱신합니다.

(3-ⓒ)에 대응하는 View함수를 참고하시면 될 것 같습니다.

ⓑ function updateUplink() / function updatePingSpeed()

```
function updateUplink() {
 $.ajax({
  url:checkUpURL,
  success:function(data) {
   $('#uplinkSpeed').text(data.upSpeed);
  },
  error:function(xhr) {
   console.log(xhr.status+': '+xhr.statusText);
  }
 });
}
function updatePingSpeed() {
 $.ajax({
  url:checkPingURL,
  success:function(data) {
   $('#pingSpeed').text(data.pingSpeed);
  },
  error:function(xhr) {
   console.log(xhr.status+': '+xhr.statusText);
  }
 });
}
```

ajax를 통해 id="uplinkSpeed"혹은 id="pingSpeed"인 태그 값을 변경하는 함수입니다. HTTP Request 및 JsonResponse 작동방식은 ⓐ와 동일합니다.

updateUplink()함수는 setInterval을 통해 25초당 한번, updatePingSpeed()는 10초에 한번씩 작동합니다.

ⓒ 채팅관련 함수

```
var csrftoken='{{ csrf_token }}';
$(document).ready(function() {
 $("#CommentSendBtn").click(function() {
  var content=document.getElementById("message_contents").value;
  var inputbox=document.getElementById("message_contents");
  $.ajax({
   url:"sendMessage/",  // 요청을 보낼 URL
   headers: { "X-CSRFToken":csrftoken },
   method:"POST",    // HTTP 메서드 (POST로 설정)
   data: {
    'message_content':content   // 전송할 데이터 (내용)
   },
   success:function(response) {
    inputbox.value='';
    // 두 번째 요청
    $.ajax({
     url:"readMessage/",  // 요청을 보낼 URL
     success:function(getData) {
      var commentList=getData.Comment_list.replace(/₩n/g, "<br>");
      $('#chattingBox').html(commentList);
     },
     error:function(xhr, status, error) {
     }
    });
   },
   error:function(xhr, status, error) {
   }
  });
 });
});
```

위 코드는 AJAX를 사용하여 채팅 메시지를 전송하고 받아오는 기능을 구현한 스크립트입니다. 문서가 로드되면 $(document).ready() 함수가 실행되고, "CommentSendBtn" 버튼이 클릭되었을 때의 동작을 정의합니다. 버튼이 클릭되면 message_contents 요소의 값을 가져와 content 변수에 저장합니다. 그리고 inputbox 변수에는 message_contents 요소를 할당합니다.

다음으로 AJAX를 사용하여 서버로 메시지를 전송합니다. 요청 URL은 "sendMessage/"이며, 전송할 데이터는 content 변수의 내용입니다. 성공적으로 요청이 완료되면 success 함수가 실행됩니다.

success 함수 내에서는 inputbox의 값을 비워줍니다. 그리고 두 번째 요청으로 AJAX를 사용하여 채팅 메시지를 받아옵니다. 요청 URL은 "readMessage/"입니다. 응답이 성공적으로 도착하면 success 함수가 실행되고, 받아온 데이터를 HTML 요소에 할당하여 채팅 내용을 업데이트합니다.

이때, 기본적으로 HTML에서는 개행문자를 무시하기 때문에, '₩n'과 같은 개행문자를 모두 HTML에서의 줄바꿈을 의미하는
로 바꾸어 적용합니다.

마지막으로, 에러 처리를 위한 코드도 포함되어 있습니다.

ⓓ Alarm Part

```
document.getElementById("alarm-form").addEventListener("submit", function(event) {
  event.preventDefault(); // 폼 제출 기본 동작 막기
  var audio=newAudio("{% static 'sound/alarm.mp3' %}");
  var alarmTimeInput=document.getElementById("alarm-time");
  var selectedRadio=document.querySelector('input[name="set_alarm"]:checked');
  var alarmTime=alarmTimeInput.value;  // 알람위젯 값 설정 위한 변수
  var alarmTimeBefore=selectedRadio.value;  // 알람위젯 값 설정 위한 변수
  alarmTimeBefore=alarmTimeBefore.toString();
  alarmTimeBefore=alarmTimeBefore+"초 전";
  var alarmNotificationElement=document.getElementById("AlarmNotification");
  alarmNotificationElement.textContent=alarmTime;
  var alarmNotificationBeforeElement=document.getElementById("AlarmNotification-Before");
  alarmNotificationBeforeElement.textContent=alarmTimeBefore;
  if (alarmTimeInput) {
   var alarmTime=alarmTimeInput.value;
   var [hours, minutes] =alarmTime.split(":"); // hours, minutes [시간, 분] , 24시간 체계
   var hoursInt=parseInt(hours, 10);
   var minutesInt=parseInt(minutes, 10);
   var deltaTime=parseInt(selectedRadio.value, 10);  // 30 or 60 or 180
   var totalInt=hoursInt*3600+minutesInt*60-deltaTime;
   var timeParts=current_time_ajax.split(" ");
   var c_hours=timeParts[3].substr(0, 2);
   var c_minutes=timeParts[4].substr(0, 2);
   var c_seconds=timeParts[5].substr(0, 2);
   var c_hoursInt=parseInt(c_hours, 10);
   var c_minutesInt=parseInt(c_minutes, 10);
   var c_secondsInt=parseInt(c_seconds, 10);
   var current_total=c_hoursInt*3600+c_minutesInt*60+c_secondsInt
   if (totalInt>current_total) {
    var timeDifference=totalInt-current_total
    timeoutID=setTimeout(function() {
     audio.play();
    }, timeDifference*1000);
   }
   else { alert("알람을 설정할 수 없습니다!"); }
  }
  else {alert("알람기준시간을 선택해주세요.");}
 });
```

위의 코드는 알람을 설정하는 기능을 구현하는 스크립트입니다. 먼저, "alarm-form" 요소의 제출 이벤트를 감지하여 기본 동작을 막습니다. 알람 소리를 재생하기 위해 "alarm.mp3" 파일 경로로부터 오디오 요소를 생성합니다. 그 다음, 알람 시간과 알람 전 시간을 가져와 해당하는 HTML 요소에 텍스트를 설정합니다.

알람 시간과 알람 전 시간이 입력된 경우, 알람 시간을 파싱하여 시간과 분으로 나눕니다. 또한 선택된 라디오 버튼에서 알람 전 시간을 가져옵니다. 이후, 시간과 분을 정수로 변환하고, 선택된 알람 전 시간을 정수로 변환합니다. 알람 시간과 알람 전 시간을 계산하여 총 시간을 구합니다.

마지막으로 현재 시간을 가져와 시간, 분, 초로 나누어 정수로 변환합니다. 현재 시간과 알람 시간의 차이를 계산하여 알람을 설정합니다. 그 결과로 도출된 알람 시간과 현재 시간의 차이를 계산하여 해당 시간이 경과한 후에 알람 소리를 재생합니다.

* 알람 시간이 현재 시간보다 과거인 경우, 알람을 설정할 수 없다는 경고창이 표시됩니다.
* 알람 시간이 입력되지 않은 경우, 알람 기준 시간을 선택하라는 경고창이 표시됩니다.

4-3. 발생한 문제 및 해결 내용

1) ms단위 표시와 실제서버시간과 오차 줄이기

이 프로젝트와 가장 유사한 웹사이트인 “네이비즘(time.navysm.net)”에서는 서버시간을 밀리초 단위로 표시해줍니다. 그렇기 때문에 이 프로젝트에서도 그게 가능하지 않을까?라고 생각해 제안서를 작성할 때 그렇게 기획하였지만, HTTP Response의 Header파일에 있는 [‘Date’]요소로는 서버시간의 밀리초 단위까지 확인하는 것은 불가능하다는 것을 깨달았습니다.

그렇다면?

네이비즘은 어떤 방식으로 작동하고 있을까?가 가장 큰 관건이었습니다. 실험을 위해, 네이비즘 사이트로 단국대학교 서버시간을 확인할 때, 최초 1회 접속 이후 인터넷 접속을 끊어보았습니다. 네이비즘 사이트는 인터넷이 없어도 작동하는 것이 불가능한데, 밀리초단위로 잘 갱신되고 있는게 이상하다고 생각되었습니다. 네이비즘 작동원리는 기업비밀이라 정보가 없지만, 자바스크립트로 클라이언트 시계에 맞춰 시간이 갱신되도록 했을거라는 생각이 들었습니다.

그렇게 하면 구현은 간편했겠지만, 제가 생각했던 ‘네이비즘 보다 정확한 서버시계’구현이 아니었기 때문에, 인터넷 지연(Internet Latency)를 계산하여 지금 화면에 표시되고 있는 서버시간이 실제 서버시간과 몇 밀리초 차이가 나는지 표시하자라는 생각에 다다랐습니다. 이를 위해선, 매 초 클라이언트 측에서 타겟URL의 서버에 HTTPRequest를 보내야 했습니다. 이를 위해서 Ajax를 채용할 수 밖에 없었습니다. 또한, AJax에서 HTTP Request를 보내는 것에서 착안하여 지난학기때 수강한 과목인 “컴퓨터 네트워크”과목에서 Persistent HTTP Request를 생각해내, 서버시간을 가져오는 시간을 더더욱 줄여 최적화에 성공하였습니다. (이를 통해, 2RTT가 실행시간이었다면, HandShaking 과정을 최초 1회 실행함으로써 서버시간을 가져오는 시간을 1RTT로 줄일수 있었습니다.)

물론, 결국은 인터넷 문제에 의해 좌지우지 되는 결과이기 때문에 인터넷 상태나 트래픽 등에 영향을 받아 오차범위가 커질 수 있다는 문제점이 존재합니다.

2) PING 정보 취득 관련

Ping은 ICMP(Internet Control Message Protocol)를 사용하여 네트워크 상의 호스트 간에 연결성과 응답 시간을 확인하기 위해 사용되는 프로토콜입니다. ICMP는 네트워크 통신에 관련된 다양한 메시지를 전송하고 수신하여 네트워크의 동작을 관리하고 진단하는 데 사용됩니다.

일반적으로 Ping은 호스트 간의 통신이 원활한지 확인하기 위해 사용되며, 목적지 호스트로 ICMP Echo Request 메시지를 보내고 해당 호스트가 이를 수신하면 ICMP Echo Reply 메시지를 반환합니다. 이를 통해 송신 호스트는 응답 시간을 측정하여 네트워크 상태를 확인할 수 있습니다.

그러나 Ping이 실패하거나 Ping Speed를 체크할 수 없는 이유는 여러 가지일 수 있습니다. 그 중 일반적인 이유는 다음과 같습니다:

ⓐ ICMP 차단: 네트워크 방화벽이 ICMP 패킷을 차단하여 ICMP 메시지가 목적지 호스트로 전달되지 않을 수 있습니다. **방화벽 설정이 ICMP Echo Request 및 Echo Reply를 차단**하는 경우, 호스트 간의 Ping 통신이 막힐 수 있습니다.

ⓑ 호스트 응답 설정: 목적지 호스트 자체에서 ICMP Echo Request를 수신하고 ICMP Echo Reply를 반환하는 것을 비활성화하는 경우, Ping 응답이 없을 수 있습니다. 이는 보안상의 이유로 호스트가 ICMP 트래픽을 제한하는 경우나 네트워크 관리 정책에 따라 발생할 수 있습니다.

ⓒ 네트워크 구성: 목적지 호스트가 올바르게 구성되어 있지 않거나 네트워크의 문제로 인해 ICMP 패킷이 목적지로 전달되지 않을 수 있습니다. 이는 네트워크 연결의 문제, 라우팅 구성 오류, 네트워크 장비의 장애 등으로 인해 발생할 수 있습니다.

이중에 눈여겨 봐야할 부분은 [ⓐ ICMP 차단] 관련입니다. 사용자가 접속하려고 하는 URL의 서버가 ICMP를 차단했는지 하지 않았는지에 따라 핑 정보를 얻을 수도 있고 없을 수도 있습니다. 단적으로, **단국대학교 수강신청 서버는 ICMP를 차단해두었다는 것을 확인**했습니다. 주로 ICMP 즉 PING을 차단해둔 이유는 트래픽 증가 혹은 DDos공격에 취약해 질 수 있기 때문입니다. 따라서, **Ping정보를 확인할 수 있는 NTP 서버(time.windows.com)에 Ping을 보내** 사용자가 이용하고 있는 인터넷의 Ping속도 정보를 얻었습니다. 하지만, 이는 타겟서버에 Ping을 보내는 것이 아니라는 의미가 되기 때문에, 타겟 서버의 트래픽을 점검할 수 없게 됩니다.

5. 테스트결과 및 분석

5-1. 전반적인 웹페이지 구성

<그림 (a)> https://localhost:8000 *firstpage.html*

<그림 (b)> https://localhost:8000/main/result *mainpage.html*

위 그림에서 확인할 수 있듯, 전반적인 화면 구성은 다음과 같습니다. 첫 화면은 welcomepage로, 사용자가 최초 웹사이트에 접속하였을 때, 나타나는 화면입니다. 입력창에 URL을 입력하고, “테스트”버튼을 클릭하면 그림(b)로 넘어와, 본 웹사이트에서 제공하는 기능을 사용할 수 있습니다.

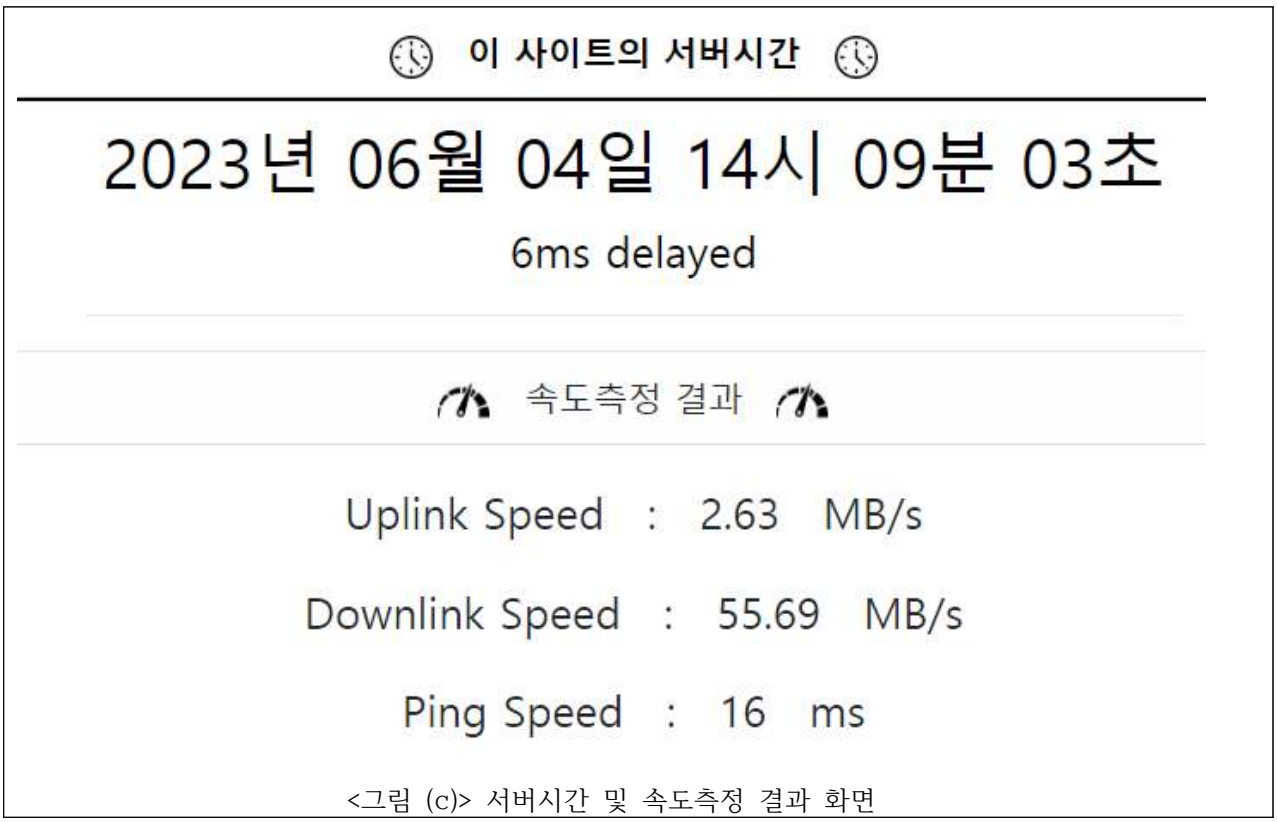
이후에는 1000밀리초 단위로 시계를 갱신하고, 속도측정 각 지표는 25~30초 간격으로 재측정됩니다. 또한, 알람기능을 통해 원하는 시간에 알람이 울릴 수 있도록 할 수 있습니다.

마지막으로, 왼쪽에 채팅창을 통해 실시간으로 흔히 ‘대기타는’사람들과 자유롭게 익명으로 채팅을 할 수 있습니다.

만약, 시간을 확인하고 싶은 사이트를 변경하고 싶다면, 위쪽의 ‘접속하고자 하는 URL을 입력해주세요’라는 플레이스홀더가 있는 input창에 사이트를 새롭게 입력하시면 됩니다.

5-2. 기능별 작동화면

1) 서버시간 & 속도측정 결과



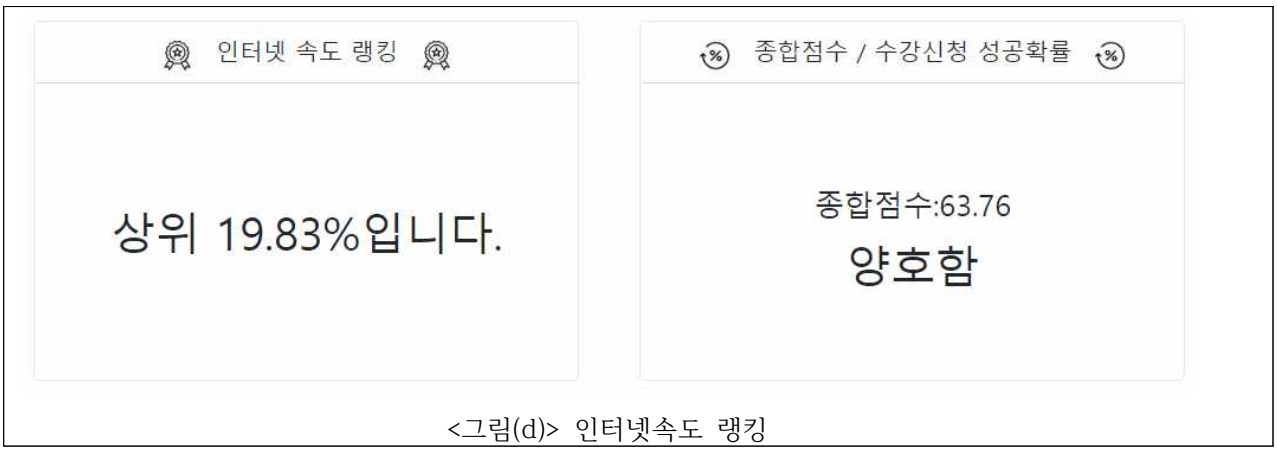
<그림 (c)> 서버시간 및 속도측정 결과 화면

<그림(c)>는 보고서를 작성하고있는 2023년 6월 4일 14이 09분 03초 기준으로, 사용자가 아무런 시간을 입력하지 않았기 때문에 Default시간(time.windows.com)[NTP서버]의 시간을 표시합니다. 이때, 시간정보를 가져오는데 걸리는 시간은 24ms이고, 실제 서버시간과의 오차는 6ms라는 것을 알 수 있습니다. 또한, 사용자(지금은 제 컴퓨터)가 이용하고 있는 인터넷 속도 정보가 출력되고 있는 것을 확인할 수 있습니다.

업로드 속도 2.63 MB/s, 다운로드 속도 55.69 MB/s, 핑 속도 :16ms

위 결과로 봤을 때, 꽤 빠른 인터넷을 사용하고 있다는 것을 확인할 수 있었습니다.

2) 인터넷 속도 랭킹 & 종합점수



<그림(d)> 인터넷속도 랭킹

그림(d)에서 볼 수 있듯, 인터넷속도 정보를 바탕으로 인터넷속도 랭킹과 종합점수가 도출된 것을 확인할 수 있습니다. 초당 50MB/s 정도의 다운로드 속도가 상위 20%에 준하는 인터넷속도라는 것을 알 수 있었습니다.

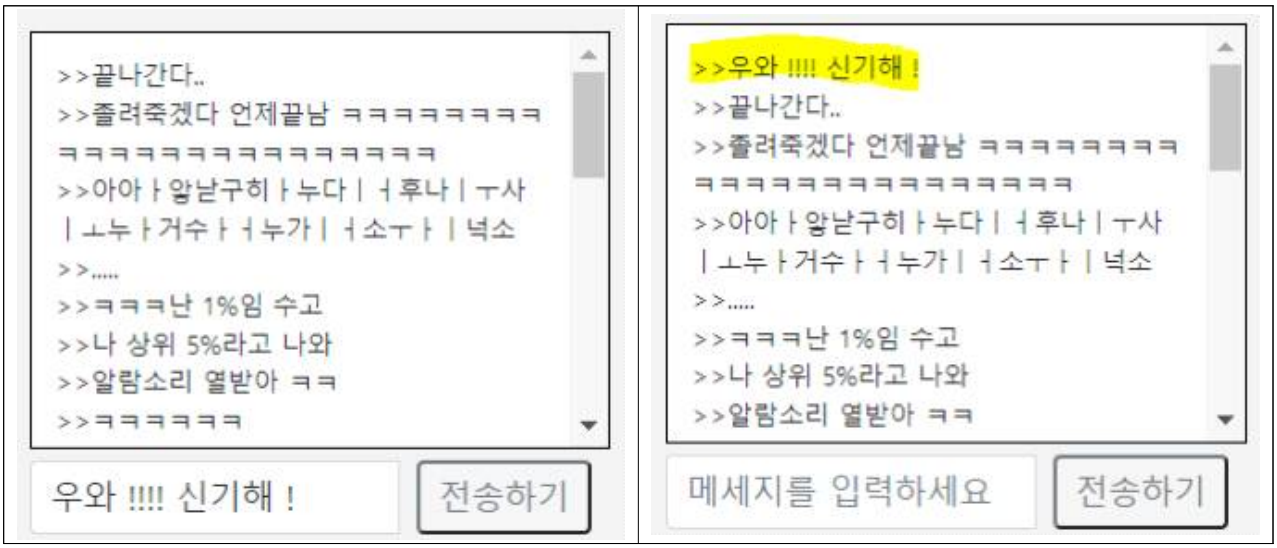
종합점수는 핑정보와 다운로드속도, 랭킹정보를 모두 포함하여 계산한 정보입니다. 이 점수를 가지고, 수강신청 성공확률을 보여주는데(물론 어디까지나 개발자의 의견입니다), 보통 정도입니다.

'나쁨' 이하가 아니라는게 다행이라고 생각됩니다.

3) 알람설정

mainpage.html의 [알람 설정하기] 위젯을 통해, 알람을 설정하면, sidbar.html의 [알람울릴시간] 위젯이 활성화되며, **15:55 180초전인 15:52:00에 알람이 울리게 됩니다.** 보고서에서는 알람이 정상적으로 작동하는 것을 보여드릴 수 없기 때문에, [발표동영상]에서 시연해 보이도록 하겠습니다.

4) 채팅 기능



sidebar.html의 채팅기능입니다. 메시지를 입력한 후 [전송하기]버튼을 클릭하면, 실시간으로 채팅에 반영된 것을 확인할 수 있었습니다. 이때, 메시지를 작성한 사람이 누구인지는 남겨지지 않습니다. DB에는 [content]와 [created_at] 두 필드만 가지고 있습니다. 하지만, DB에는 작성자의 ip가 남게됩니다.

5) 정보 확인 버튼

표의 왼쪽사진의 각 버튼을 클릭하면, 오프캔버스가 작동합니다. 사용자가 궁금해 할 만한 정보를 보여줄 수 있게 됩니다. 오른쪽 사진은, [속도랭킹 & 성공확률 정보] 버튼을 클릭했을 때 팝업되는 오프캔버스 내용입니다.

6) 광고

보고서를 작성하기 막 작성하기 시작했을 때엔, [광고] 부분은 “이 부분은 광고가 표시됩니다”로 남겨두었었는데, 비워두기엔 아쉬워서 일단 단국대학교 로고를 두 개 삽입해 보았습니다. 광고로 표시되는 배너(로고)가 적당한 시간차를 두고 자동으로 바뀌게 하기 위해, 부트스트랩의 “캐러샐” 요소를 사용하였습니다. 위 배너를 클릭하면, 새 탭이 열리면서 “https://dankook.ac.kr/”로 접속하게 됩니다. 차후, 실제 서비스를 배포하게 된다면 이 부분은 구글 adsense등을 통하여 수익성을 내보려고 합니다.

7) 푸터

마지막으로, 개발자 정보를 담고 있는 푸터정보입니다. 제 이메일 주소와, 학번&이름, 깃헙주소가 포함되어있습니다. github.com/hckang17 부분을 클릭하면 제 깃헙으로 이동하게 됩니다.

6. 후기

이번 프로젝트에서는 서버시간을 보여주는 웹페이지를 만드는 작업을 진행했습니다. 전반적으로 프로젝트는 성공적으로 완료되었고, 몇 가지 주요한 측면에서 개인적으로 매우 만족스러운 결과물을 얻을 수 있었습니다.

첫째로, 기술적인 면에서의 성장이 있었습니다. 프로젝트를 진행하면서 웹 개발과 관련된 지식인 HTML, 그리고 JavaScript에 대한 이해도를 비약적으로 높일 수 있었습니다. Django라는 웹프레임워크를 이용하여 개발을 진행하면서, Python를 다루는 스킬 또한 비약적으로 성장하였습니다. 또한, “서버시간”을 받아오는 함수를 최적화 하는 과정에서, 네트워크와 관련된 프로토콜에 대한 복습도 할 수 있었습니다. 여기서, OSI 7계층 및 실제 네트워크상에서 어떤 규약(프로토콜)을 가지고 통신을 수행하는지에 대해 알게되었습니다. 특히, Transport Layer의 두 프로토콜인 TCP와 UDP의 성질에 대해 다시한번 공부하고, Application Layer단계에서 통신관련 API를 통해 하위 계층을 다루는 방법에 대해서도 공부하게 되었습니다.

둘째로, 이번 프로젝트를 통해 풀스택 개발을 진행하면서 가장 힘들었던 점은 프론트엔드 쪽 개발이었습니다. Django와 Bootstrap 웹프레임워크를 가지고 웹페이지를 구성한다고 해도, Django는 비동기적인 HTML갱신이 되지 않았기 때문에, 결국엔 자바스크립트에 대한 의존도가 많이 올라갈 수 밖에 없었습니다. 또한, HTML 구성요소를 작성할 때 제가 의도했던 대로 CSS 스타일링이 적용되지 않는 경우가 많아, 이 부분을 고치기 위해 많은 시간을 쓸 수 밖에 없었기 때문입니다.

그럼에도 불구하고, 실제로 동작하는 서버시간을 보여주는 웹페이지를 완성하게 되어 매우 뿌듯합니다. 기존에 서비스 하고 있는 네이비즘이라는 웹페이지보다 더 정밀한 서버시간을 표시할 수 있는 시계를 구현했다는 것이 너무 뿌듯했습니다. 2023학년 2학기 수강신청때는 제가 만든 서버시계를 통해 수강신청을 시도해볼 생각입니다.

마지막으로, 이번 프로젝트를 진행할 수 있도록 좋은 수업을 해주신 송인식 교수님에게 감사인사 드립니다.

7. 참고문헌

1. Internet Latency, Packet Delay
https://www.kentik.com/kentipedia/understanding-latency-packet-loss-and-jitter-in-networking/

2. Computer Network
Professor, Suhan Choi's Lecture Note.

3. TCP/IP Protocol
https://www.ibm.com/docs/ko/aix/7.1?topic=protocol-tcpip-protocols

4. Ajax Query
https://api.jquery.com/jquery.ajax/

5. Bootstrap
https://getbootstrap.kr/

6. Django Documents
https://docs.djangoproject.com/en/4.2/

7. Python Library(Speedtest_cli) Documents
https://pypi.org/project/speedtest-cli/