**POST TEST**

**PEMROGRAMAN WEB 1**

**MODUL 7**



NAMA : Aef Nur Hidayah

NIM : 20230910054

KELAS : SINFC – 2023 - 03

FAKULTAS ILMU KOMPUTER

UNIVERSITAS KUNINGAN

TAHUN 2024

# **PRETEST**

1. Apa prinsip utama dari arsitektur REST dan bagaimana prinsip tersebut mempengaruhi desain API?

Arsitektur **REST** (Representational State Transfer) punya beberapa prinsip utama yang berpengaruh besar dalam desain API. Pertama, prinsip **Client-Server** memastikan klien dan server terpisah, jadi keduanya bisa dikembangkan secara mandiri tanpa saling bergantung. Kedua, prinsip **Stateless** mengharuskan setiap permintaan dari klien membawa semua data yang dibutuhkan, tanpa server menyimpan sesi, sehingga lebih simpel dan mudah diskalakan. Ketiga, REST mendukung **Caching**, di mana data dari server bisa disimpan sementara di klien agar permintaan berikutnya lebih cepat dan tidak membebani server. Keempat, REST wajib punya **Uniform Interface**, jadi semua resource diakses lewat URL (URI), pakai HTTP methods kayak GET, POST, PUT, atau DELETE, dan hasilnya biasanya berupa JSON atau XML dengan status code yang jelas. Selain itu, REST juga pakai prinsip **Layered System**, jadi bisa pakai middleware seperti cache atau load balancer tanpa bikin klien tahu detail implementasi server. Prinsip opsional lainnya adalah **Code on Demand**, di mana server bisa ngasih kode tambahan ke klien untuk menjalankan fungsi tertentu. Intinya, prinsip-prinsip ini bikin API REST jadi lebih sederhana, terstruktur, gampang dipakai, dan fleksibel untuk diintegrasikan di berbagai sistem.

1. Jelaskan perbedaan antara metode HTTP GET dan POST dalam konteks API

Metode HTTP **GET** dan **POST** adalah dua metode utama yang sering digunakan dalam API, dan masing-masing memiliki fungsi serta karakteristik yang berbeda.

1. **GET** digunakan untuk mengambil data dari server. Metode ini bersifat *read-only* dan tidak mengubah data di server. Data yang diminta dikirim melalui URL sebagai parameter query string (misalnya, ?id=123). Karena itu, **GET** cocok untuk operasi seperti membaca data (misalnya, mendapatkan detail produk atau daftar pengguna). Selain itu, **GET** dapat di-cache oleh browser, hasilnya bisa dibookmark, dan lebih mudah untuk diuji karena parameternya terlihat langsung di URL. Namun, **GET** memiliki batasan ukuran pada URL, sehingga tidak cocok untuk mengirim data yang besar.
2. **POST**, di sisi lain, digunakan untuk mengirimkan data ke server untuk diproses, seperti membuat data baru atau mengirim formulir. Data yang dikirim melalui **POST** tidak ditampilkan di URL melainkan di *body* permintaan HTTP, sehingga lebih aman untuk data sensitif dibandingkan **GET**. Metode ini tidak bisa di-cache oleh browser, tidak bisa dibookmark, dan sering digunakan untuk operasi yang membutuhkan perubahan di server, seperti pendaftaran akun atau pengiriman pesan.
3. Apa fungsi SpreadsheetApp dalam Google Apps Script dan bagaimana cara mengakses data dari Google Sheets.

**SpreadsheetApp** dalam **Google Apps Script** adalah layanan bawaan yang digunakan untuk mengelola dan memanipulasi spreadsheet di Google Sheets. Dengan SpreadsheetApp, kita bisa membuat, membaca, memperbarui, dan menghapus data dari spreadsheet, serta mengatur format, formula, dan properti lainnya.

**Fungsi Utama SpreadsheetApp**

1. **Mengakses Spreadsheet**:
   * Membuka spreadsheet yang ada.
   * Membuat spreadsheet baru.
   * Mendapatkan spreadsheet aktif yang sedang digunakan oleh skrip.
2. **Memanipulasi Data**:
   * Membaca data dari sel, baris, atau kolom.
   * Menulis atau mengubah data dalam spreadsheet.
   * Menghapus atau memformat data.
3. **Mengelola Struktur**:
   * Menambahkan atau menghapus sheet.
   * Mengubah nama sheet.
   * Menyusun ulang sheet dalam spreadsheet.

Untuk mengakses data dari Google Sheets menggunakan **SpreadsheetApp** di Google Apps Script, pertama-tama dapatkan spreadsheet dengan SpreadsheetApp.getActiveSpreadsheet() untuk spreadsheet aktif, atau gunakan SpreadsheetApp.openById("SPREADSHEET\_ID") jika memiliki ID spreadsheet. Setelah itu, pilih sheet yang ingin diakses menggunakan spreadsheet.getSheetByName("NamaSheet") atau spreadsheet.getSheets()[0] untuk sheet pertama. Gunakan metode getRange("A1:B10").getValues() untuk membaca data dalam bentuk array 2D dari rentang tertentu, atau getRange("A1").getValue() untuk membaca satu sel. Untuk menulis data, gunakan setValue("Nilai") untuk satu sel, atau setValues(arrayData) untuk rentang data. Semua proses ini memungkinkan manipulasi data di Google Sheets secara efisien.

1. Sebutkan 3 komponen utama dalam struktur Google Sheets dan jelaskan fungsinya.

3 komponen utama dalam struktur Google Sheets beserta fungsinya:

1. **Spreadsheet**
   * **Fungsi**: Merupakan file utama yang menyimpan semua data dan pengaturan. Spreadsheet dapat memiliki beberapa sheet (tab) dan berisi berbagai informasi yang dikelompokkan dalam baris dan kolom. Di Google Apps Script, spreadsheet diakses menggunakan SpreadsheetApp.
2. **Sheet**
   * **Fungsi**: Merupakan tab individual dalam sebuah spreadsheet yang menyimpan data dalam format tabel. Setiap sheet memiliki nama unik dan berisi baris serta kolom. Sheet digunakan untuk memisahkan atau mengorganisasi data ke dalam kategori yang berbeda. Dalam skrip, sheet dapat diakses dengan spreadsheet.getSheetByName("NamaSheet").
3. **Range**
   * **Fungsi**: Merupakan blok data tertentu yang terdiri dari satu atau lebih sel dalam sheet. Range digunakan untuk membaca, menulis, atau memanipulasi data. Range dapat diakses menggunakan sheet.getRange("A1:B10") untuk rentang tertentu atau sheet.getRange(row, column) untuk satu sel.

Komponen-komponen ini bekerja bersama untuk mengatur dan memanipulasi data di Google Sheets.

1. Apa itu JSON dan bagaimana formatnya digunakan dalam REST API untuk mengirim data?

**JSON** (JavaScript Object Notation) adalah format ringan untuk pertukaran data yang mudah dibaca manusia dan mesin. JSON sering digunakan dalam REST API karena sifatnya yang sederhana, fleksibel, dan kompatibel dengan berbagai bahasa pemrograman.

JSON menyimpan data dalam bentuk pasangan **key-value**, mirip dengan objek dalam JavaScript. Formatnya menggunakan kurung kurawal {} untuk menandai objek dan tanda kurung siku [] untuk daftar (array). Contoh format JSON:

{

"nama": "Aef Nur Hidayah",

"umur": 20,

"hobi": ["coding", "membaca", "musik"],

"isMahasiswa": true

}

* **Key** adalah string yang dikelilingi tanda kutip ganda (" ").
* **Value** bisa berupa tipe data seperti string, angka, boolean, array, objek lain, atau null.

1. Langkah – Langkah apa yang diperlukan untuk membuat dan menjalankan script Google Apps Script yang dapat digunakan sebagai endpoint API?

**1. Buat Proyek Apps Script**

1. Buka [Google Apps Script Editor](https://script.google.com/).
2. Klik **New Project** untuk membuat proyek baru.

**2. Tulis Fungsi Utama dalam Script**

* Tulis fungsi yang akan menangani permintaan API. Gunakan doGet(e) untuk menangani permintaan GET dan doPost(e) untuk POST. Contoh skrip:

function doGet(e) {

return ContentService.createTextOutput("Hello, world!").setMimeType(ContentService.MimeType.TEXT);

}

function doPost(e) {

let data = JSON.parse(e.postData.contents);

let response = {

message: "Data received",

receivedData: data

};

return ContentService.createTextOutput(JSON.stringify(response)).setMimeType(ContentService.MimeType.JSON);

}

**3. Publikasikan Proyek sebagai Web App**

1. Klik menu **Deploy** di bagian atas editor dan pilih **New deployment**.
2. Pada bagian **Select type**, pilih **Web app**.
3. Isi deskripsi (opsional) dan tentukan:
   * **Execute as**: Pilih **Me** untuk menjalankan skrip dengan kredensial Anda.
   * **Who has access**: Pilih **Anyone** jika ingin endpoint dapat diakses publik, atau **Anyone with Google account** untuk lebih terbatas.
4. Klik Deploy.

### **4. Salin URL Web App**

### Setelah deploy, salin URL yang diberikan. URL ini adalah endpoint API Anda, misalnya:

<https://script.google.com/macros/s/AKfycbyX.../exec>

5. **Uji Endpoint API**

Gunakan tools seperti **Postman**, **cURL**, atau browser untuk menguji endpoint:

* **GET**: Buka URL endpoint di browser atau gunakan cURL:

curl -X GET <https://script.google.com/macros/s/AKfycbyX.../exec>

* **POST**: Kirim permintaan POST dengan data JSON menggunakan cURL:

curl -X POST "https://script.google.com/macros/s/AKfycbyX.../exec" \

-H "Content-Type: application/json" \

-d '{"name": "Aef", "age": 20}'

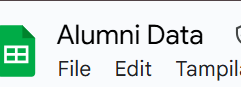
6. **Perbaiki dan Iterasi Skrip**

Jika ada perubahan, edit skrip di Apps Script Editor, lalu:

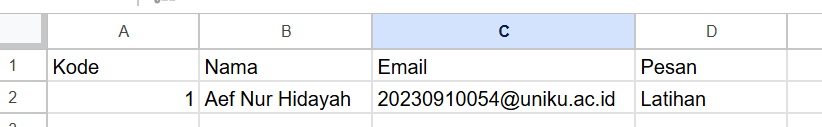
1. Simpan perubahan.
2. Klik **Deploy** > **Manage deployments**.
3. Perbarui deployment untuk memastikan endpoint menggunakan skrip terbaru.

# **B. PRAKTIKUM**

1. Buat speedsheet baru di google drive Bernama “Alumni Data”

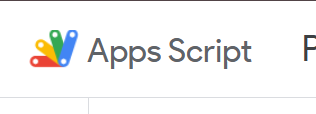


2. Buat sheet BukuTamu yang berisi kolom Kode, Nama, Email, Pesan.



Membuat API di Google Apps Script

* 1. Buka Google Apps Script dengan cara pada Google Spreadsheet, klik menu Extension kemudian pilih Apps Script.



* 1. Ketikan code berikut pada editor Apps Script :



* 1. Deploy sebagai Web Apps

<https://script.google.com/a/macros/uniku.ac.id/s/AKfycbwpws3xVlvj57Wx4KqSCopCmt1fHeF_jn45WMJNO6APXOzmDuwgOlcdU9MiCJOO6povzg/exec>

Membuat FrontEnd

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Buku Tamu CRUD</title>      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">      <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>      <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>      <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>  </head>  <body>      <div class="container mt-5">          <h2>Buku Tamu</h2>          <form id="guestForm" class="mb-4">              <div class="form-group">                  <label for="guestKode">Kode:</label>                  <input type="text" class="form-control" id="guestKode" required>              </div>              <div class="form-group">                  <label for="guestNama">Nama:</label>                  <input type="text" class="form-control" id="guestNama"></label>              </div>              <div class="form-group">                  <label for="guestEmail">Email:</label>                  <input type="text" class="form-control" id="guestEmail"></label>              </div>              <div class="form-group">                  <label for="guestPesan">Pesan:</label>                  <input type="text" class="form-control" id="guestPesan"></label>              </div>              <button type="submit" class="btn btn-primary" id="submitBtn">Add Guests</button>          </form>          <!-- Table to Display -->           <table class="table table-striped">              <thead>                  <tr>                      <th>Kode</th>                      <th>Nama</th>                      <th>Email</th>                      <th>Pesan</th>                      <!-- <th>Actions</th> -->                  </tr>              </thead>              <!-- Guest rows will be appended here -->           </table>      </div>     <script>          $(document).ready(function(){              const script\_url = 'https://script.google.com/a/macros/uniku.ac.id/s/AKfycbwpws3xVlvj57Wx4KqSCopCmt1fHeF\_  jn45WMJNO6APXOzmDuwgOlcdU9MiCJOO6povzg/exec';              //load guests on page load              loadGuests();              $("#guestForm").on('submit',function(event){                  event.preventDefault();                  const kode = $("#guestKode").val();                  const nama = $("#guestNama").val();                  const email = $("#guestEmail").val();                  const pesan = $("#guestPesan").val();                  $.ajax({                      url: script\_url,                      type: 'GET',                      data: { kode:kode,nama:nama,email:email,pesan:pesan,action : 'insert' },                      success: function(response){                          alert(response.result);                          loadGuests();                      },                      error: function(jqXHR, textStatus, errorThrown){                          console.error("Request failed: " + textStatus + ", " + errorThrown);                      }                  });              });              //load guests into the table             function loadGuests(){                  $.ajax({                      url: script\_url,                      type: 'GET',                      data: { action : "read" };                      dataType: 'json',                      success: function(response){                          const tbody = $('#guestTableBody');                          tbody.empty();                          response.forEach(guest => {                              tbody.append(`                                  <tr>                                      <td>${guest.Kode}</td>                                      <td>${guest.Nama}</td>                                      <td>${guest.Email}</td>                                      <td>${guest.Pesan}</td>                                  </tr>                              `);                          });                      },                      error: function(jqXHR, textStatus, errorThrown){                          console.error("Request failed: " + textStatus + ", " + errorThrown);                      }                  });              }          });      </script>  </body>  </html> |

Membuat Form untuk menambahkan data

|  |
| --- |
| <form id="guestForm" class="mb-4">              <div class="form-group">                  <label for="guestKode">Kode:</label>                  <input type="text" class="form-control" id="guestKode" required>              </div>              <div class="form-group">                  <label for="guestNama">Nama:</label>                  <input type="text" class="form-control" id="guestNama"></label>              </div>              <div class="form-group">                  <label for="guestEmail">Email:</label>                  <input type="text" class="form-control" id="guestEmail"></label>              </div>              <div class="form-group">                  <label for="guestPesan">Pesan:</label>                  <input type="text" class="form-control" id="guestPesan"></label>              </div>              <button type="submit" class="btn btn-primary" id="submitBtn">Add Guests</button>          </form> |

Penambahan code javascript

|  |
| --- |
| $("#guestForm").on('submit',function(event){                  event.preventDefault();                  const kode = $("#guestKode").val();                  const nama = $("#guestNama").val();                  const email = $("#guestEmail").val();                  const pesan = $("#guestPesan").val();                  $.ajax({                      url: script\_url,                      type: 'GET',                      data: { kode:kode,nama:nama,email:email,pesan:pesan,action : 'insert' },                      success: function(response){                          alert(response.result);                          loadGuests();                      },                      error: function(jqXHR, textStatus, errorThrown){                          console.error("Request failed: " + textStatus + ", " + errorThrown);                      }                  });              }); |

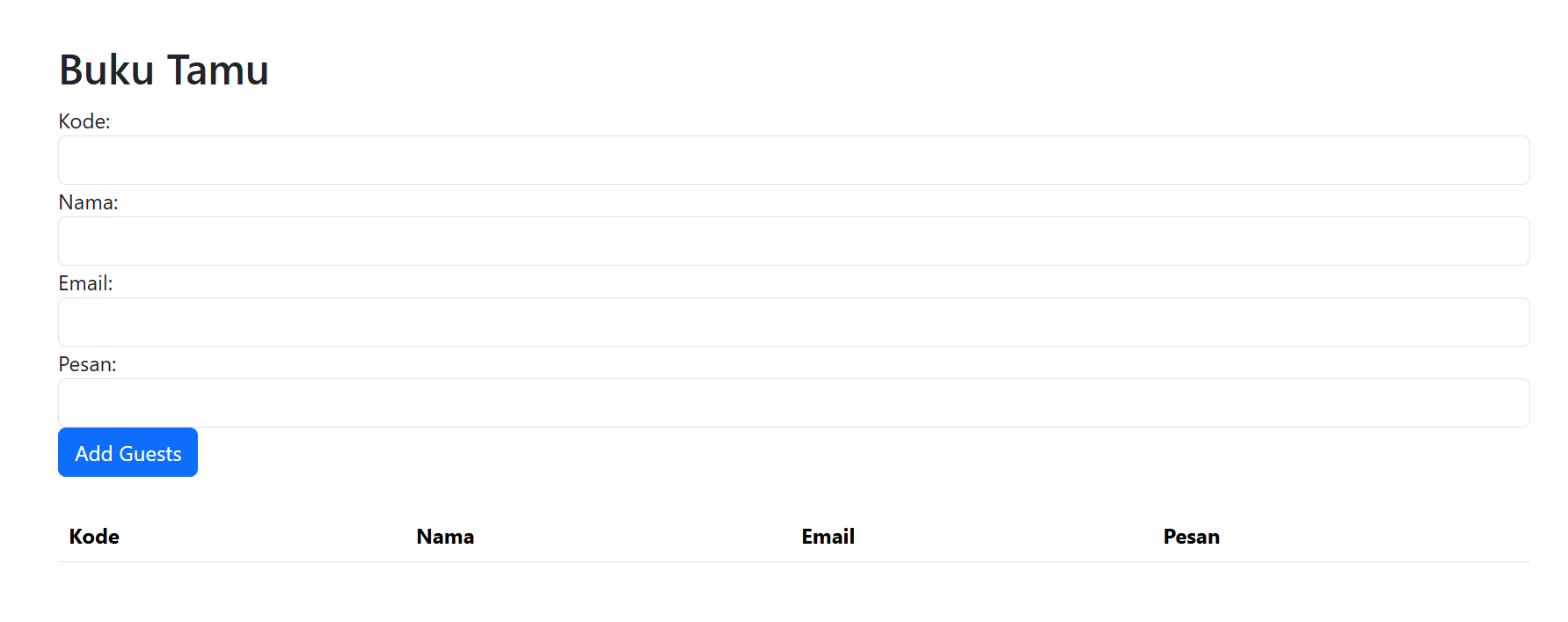
Perubahan doGet() pada Apps Script

|  |
| --- |
| if (op == "read") {      return read();    } else if (op == "insert") {      return insert\_value(e);    } |

Menambahkan fungsi baru

|  |
| --- |
| function insert\_value(request) {    var wb = SpreadsheetApp.getActiveSpreadsheet();    var ws = wb.getSheetByName('BukuTamu');      if (!ws) {      return ContentService.createTextOutput("Error: Sheet 'BukuTamu' tidak ditemukan")        .setMimeType(ContentService.MimeType.TEXT);    }    var kode = request.parameter.kode;    var nama = request.parameter.nama;    var email = request.parameter.email;    var pesan = request.parameter.pesan;    if (!kode || !nama || !email || !pesan) {      return ContentService.createTextOutput("Error: Semua parameter harus diisi (kode, nama, email, pesan)")        .setMimeType(ContentService.MimeType.TEXT);    }    ws.appendRow([kode, nama, email, pesan]);    var result = "Insert successful";    result = JSON.stringify({      "result": result    });    if (request.parameter.callback) {      return ContentService        .createTextOutput(request.parameter.callback + "(" + result + ")")        .setMimeType(ContentService.MimeType.JAVASCRIPT);    }    return ContentService      .createTextOutput(result)      .setMimeType(ContentService.MimeType.JSON);  } |

Hasil :



# **C. POST TEST**

1. Jelaskan bagaimana anda akan mendesign API untuk mengelola data inventaris menggunakan Google Script dan GoogleSheets.

Untuk merancang API yang mengelola data inventaris menggunakan Google Script dan Google Sheets, pertama-tama kita perlu menentukan struktur data di Google Sheets, seperti membuat kolom untuk ID, nama barang, deskripsi, jumlah, harga, dan lokasi. Data ini kemudian akan disimpan di dalam sheet yang bisa kita akses menggunakan Google Apps Script. Selanjutnya, kita akan membuat fungsi-fungsi di Google Apps Script untuk menangani permintaan CRUD (Create, Read, Update, Delete). Fungsi doGet akan berfungsi sebagai titik masuk untuk menerima permintaan HTTP yang berbeda, seperti mengambil data inventaris, menambahkan item baru, memperbarui data barang yang ada, dan menghapus item.

Misalnya, untuk mengambil data inventaris, kita bisa membuat fungsi readInventory yang mengakses semua baris di Google Sheets dan mengembalikannya dalam format JSON. Untuk menambah item, fungsi createItem akan menerima parameter data melalui URL dan menambahkannya sebagai baris baru di Google Sheets. Fungsi updateItem akan mencari item berdasarkan ID dan memperbarui atributnya, sementara deleteItem akan menghapus baris berdasarkan ID. Setiap fungsi akan mengembalikan respons yang sesuai agar pengguna bisa mengetahui apakah operasi berhasil atau tidak.

Setelah itu, API bisa diakses melalui URL yang disediakan oleh Google Apps Script. Setiap permintaan akan diproses oleh fungsi yang sesuai, dan data inventaris akan diperbarui di Google Sheets. Untuk menjaga keamanan API, kita bisa menambahkan otentikasi, seperti menggunakan Google OAuth atau API key, untuk membatasi akses ke API tersebut. Dengan pendekatan ini, API dapat digunakan untuk mengelola inventaris secara otomatis melalui Google Sheets tanpa perlu database terpisah, yang sangat praktis untuk aplikasi kecil atau menengah.

2. Apa perbedaan utama dalam penanganan request GET dan POST pada API yang dibangun dengan Google Apps Script?

Perbedaan utama dalam penanganan request GET dan POST pada API yang dibangun dengan Google Apps Script terletak pada cara data dikirimkan dan tujuan penggunaan masing-masing metode.

1. **GET Request**:
   * **Tujuan**: Biasanya digunakan untuk **mengambil data** atau **membaca informasi** dari server. Dalam konteks API yang menggunakan Google Apps Script, GET request digunakan untuk mengambil data dari Google Sheets, misalnya untuk membaca daftar inventaris.
   * **Cara Pengiriman Data**: Data biasanya dikirimkan melalui parameter di URL. Misalnya, ketika kita ingin mengambil data inventaris, kita bisa mengirimkan parameter action=read dalam URL, seperti ?action=read. Parameter lainnya, seperti ID atau nama barang, juga bisa ditambahkan sebagai bagian dari query string.
   * **Tidak ada perubahan data**: GET tidak digunakan untuk mengubah data di server. Ini hanya untuk pengambilan informasi tanpa merubah status data yang ada.
2. **POST Request**:
   * **Tujuan**: Digunakan untuk **mengirimkan data** ke server dengan tujuan untuk **memodifikasi** atau **menambah** data. Dalam API yang dibangun dengan Google Apps Script, POST request sering digunakan untuk menambah data baru ke Google Sheets atau memperbarui data yang sudah ada.
   * **Cara Pengiriman Data**: Data dikirimkan di dalam **body** permintaan, bukan sebagai bagian dari URL. Ini memungkinkan pengiriman data yang lebih kompleks atau lebih besar (seperti objek JSON atau form data) tanpa batasan panjang URL.
   * **Mengubah status data**: POST digunakan ketika ada perubahan pada data yang ada di server, seperti menambah, memperbarui, atau menghapus informasi.

3. Berikan contoh praktis dari penggunaan Google Sheets sebagai database dalam API, dan diskusikan kelebihan serta keterbatasannya.

**1. Menyiapkan Google Sheets**

Buat spreadsheet baru di Google Sheets dengan kolom-kolom berikut:

* **ID**: ID unik untuk setiap barang.
* **Nama**: Nama barang.
* **Deskripsi**: Deskripsi barang.
* **Jumlah**: Jumlah barang yang tersedia.
* **Harga**: Harga barang.
* **Lokasi**: Lokasi barang.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Nama | Deskripsi | Jumlah | Harga | Lokasi |
| 1 | Laptop | Dell Inspiron | 10 | 5000 | Gudang A |
| 2 | Printer | Canon PIXMA | 15 | 1000 | Gudang B |

**2. Menulis Google Apps Script**

Buka Google Sheets dan buka **Extensions > Apps Script**. Di dalam editor, buat script berikut untuk menangani permintaan API menggunakan GET dan POST.

|  |
| --- |
| function doGet(e) {      const action = e.parameter.action;      if (action === 'read') {        return readData();      } else {        return ContentService.createTextOutput('Action not recognized');      }    }      function doPost(e) {      const action = e.parameter.action;      if (action === 'create') {        return createData(e.parameter);      } else if (action === 'update') {        return updateData(e.parameter);      } else {        return ContentService.createTextOutput('Action not recognized');      }    }     function readData() {      const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName('Inventaris');      const data = sheet.getDataRange().getValues();      const jsonData = [];        // Skip the header row and convert to JSON format      for (let i = 1; i *<* data.length; i++) {        jsonData.push({          ID: data[i][0],          Nama: data[i][1],          Deskripsi: data[i][2],          Jumlah: data[i][3],          Harga: data[i][4],          Lokasi: data[i][5]        });      }        return ContentService.createTextOutput(JSON.stringify(jsonData)).setMimeType  (ContentService.MimeType.JSON);    }    function createData(params) {  const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName('Inventaris');  sheet.appendRow([params.id, params.nama, params.deskripsi, params.jumlah,  params.harga, params.lokasi]);  return ContentService.createTextOutput('Data successfully added');  }    function updateData(params) {  const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName('Inventaris');  const rows = sheet.getDataRange().getValues();  let updated = false;    // Find row by ID and update data  for (let i = 1; i < rows.length; i++) {  if (rows[i][0] === params.id) {  sheet.getRange(i + 1, 2).setValue(params.nama);  sheet.getRange(i + 1, 3).setValue(params.deskripsi);  sheet.getRange(i + 1, 4).setValue(params.jumlah);  sheet.getRange(i + 1, 5).setValue(params.harga);  sheet.getRange(i + 1, 6).setValue(params.lokasi);  updated = true;  break;  }  }    return updated ? ContentService.createTextOutput('Data successfully updated') :  ContentService.createTextOutput('ID not found');  } |

**3. Menjalankan API**

* **GET Request** untuk mengambil data inventaris:

|  |
| --- |
| https://script.google.com/macros/s/{YOUR\_SCRIPT\_ID}/exec?action=read |

* **POST Request** untuk menambah data inventaris:

|  |
| --- |
| POST https://script.google.com/macros/s/{YOUR\_SCRIPT\_ID}/exec  Body:  {  "action": "create",  "id": "3",  "nama": "Kabel Ethernet",  "deskripsi": "Kabel 10m",  "jumlah": "50",  "harga": "100",  "lokasi": "Gudang C"  } |

* **POST Request** untuk memperbarui data inventaris:

|  |
| --- |
| POST https://script.google.com/macros/s/{YOUR\_SCRIPT\_ID}/exec  Body:  {  "action": "update",  "id": "3",  "nama": "Kabel Ethernet",  "deskripsi": "Kabel 20m",  "jumlah": "30",  "harga": "150",  "lokasi": "Gudang D"  } |

**4. Kelebihan Menggunakan Google Sheets sebagai Database:**

1. **Kemudahan Akses**: Google Sheets mudah diakses dan dikelola. Tidak perlu pengaturan server atau database terpisah.
2. **Gratis**: Google Sheets menawarkan kapasitas penyimpanan gratis yang cukup besar, cukup untuk aplikasi skala kecil hingga menengah.
3. **Kolaborasi**: Data dapat dengan mudah dibagikan dan diakses oleh tim lain dengan izin kolaborasi di Google Sheets.
4. **Integrasi dengan Google Apps**: Dapat terintegrasi dengan produk Google lainnya, seperti Google Drive dan Google Forms.

**5. Keterbatasan Menggunakan Google Sheets sebagai Database:**

1. **Batasan Kinerja**: Google Sheets bukan database relasional dan memiliki batasan dalam jumlah baris (hingga 10 juta sel per spreadsheet) dan kecepatan pemrosesan. Untuk aplikasi dengan data yang sangat besar atau kompleks, kinerja bisa menjadi masalah.
2. **Keterbatasan Fitur**: Google Sheets tidak memiliki fitur canggih seperti indexing, query yang kompleks, atau transaksi ACID yang dimiliki oleh database relasional.
3. **Keamanan**: Walaupun Google Sheets dapat dibatasi aksesnya, data yang sensitif mungkin lebih baik disimpan dalam sistem yang lebih aman dengan kontrol akses yang lebih ketat.
4. **Fleksibilitas**: Google Sheets terbatas dalam hal fleksibilitas query dibandingkan dengan menggunakan database seperti MySQL atau PostgreSQL.

4. Bagaimana anda akan menangani error atau status code yang berbeda dalam response API?

Setiap jenis response yang dikirimkan oleh server sebaiknya disertai dengan status code yang menggambarkan hasil dari request yang dilakukan. Sebagai contoh, jika request berhasil diproses, server bisa mengirimkan status code 200 (OK) atau 201 (Created) dengan informasi terkait dalam body response. Namun, jika ada kesalahan seperti data yang tidak valid, status code 400 (Bad Request) bisa digunakan, disertai dengan pesan yang menjelaskan kesalahan tersebut. Selain itu, jika terjadi kesalahan pada server, seperti masalah dalam pemrosesan data, status code 500 (Internal Server Error) lebih tepat digunakan.

Google Apps Script memungkinkan kita untuk mengatur status code dengan menggunakan setStatusCode() pada response, sehingga kita dapat memberi informasi yang tepat kepada klien tentang hasil dari request yang dilakukan. Misalnya, jika parameter yang dikirimkan tidak lengkap, kita bisa mengembalikan status code 400 dan memberi pesan yang jelas bahwa data yang dikirimkan tidak valid. Di sisi lain, jika ada kesalahan pada server, kita bisa memberikan status code 500 dengan penjelasan lebih lanjut mengenai masalah yang terjadi.

Selain mengatur status code, penting juga untuk memberikan detail error dalam body response. Hal ini akan memudahkan pengembang atau pengguna API dalam memahami apa yang menyebabkan kegagalan atau kesalahan dalam request. Contohnya, jika parameter yang diperlukan tidak ada, kita bisa mengirimkan response dengan status code 400 dan menjelaskan bahwa parameter tersebut wajib diisi. Sebaliknya, jika berhasil, kita bisa memberikan informasi yang jelas tentang hasil request dengan status code 200 atau 201. Penanganan error yang baik ini membantu aplikasi klien untuk menangani berbagai kondisi dan memberikan pengalaman pengguna yang lebih baik.

5. Apa Langkah – Langkah yang anda ambil untuk menguji endpoint API yang dibuat dengan Google Apps Script dan Google Sheet, dan alat apa yang akan anda gunakan?

Langkah pertama adalah memastikan bahwa script Google Apps telah berhasil di-deploy sebagai web app dan dapat diakses melalui URL yang disediakan. Hal ini penting karena endpoint API hanya dapat diuji setelah script tersebut aktif dan dapat menerima request dari klien.

Setelah itu, langkah kedua adalah memverifikasi bahwa Google Sheets yang digunakan sebagai database sudah terhubung dengan benar dan data yang diperlukan telah disiapkan di dalamnya. Pastikan kolom dan baris yang digunakan oleh script sesuai dengan struktur data yang diharapkan agar API dapat berfungsi dengan baik. Kemudian, langkah selanjutnya adalah menguji API dengan menggunakan alat seperti Postman atau curl untuk mengirimkan request GET, POST, PUT, atau DELETE ke endpoint yang telah disediakan oleh script. Dengan Postman, kita dapat dengan mudah mengirim request dengan berbagai metode HTTP dan melihat status code serta response body yang diterima dari server.

Selama pengujian, penting untuk memeriksa apakah API memberikan status code yang sesuai, seperti 200 untuk request yang berhasil atau 400 untuk request yang mengandung kesalahan. Selain itu, perhatikan apakah response body berisi data yang sesuai atau pesan error yang informatif jika terjadi masalah. Jika perlu, kita juga bisa menggunakan alat pengujian otomatis seperti Jest atau Mocha untuk melakukan unit test pada API dan memverifikasi fungsionalitas endpoint secara lebih mendalam. Dengan langkah-langkah tersebut, kita bisa memastikan bahwa API yang dibangun berjalan dengan baik, memberikan response yang sesuai, dan dapat menangani berbagai kondisi yang mungkin terjadi.

# **D. LATIHAN / TUGAS**

1. Buat API menggunakan Google Apps Script yang memungkinkan pengelolaan lowongan kerja dalam Google Sheets. API harus mencakup kemampuan untuk menambahkan, memperbarui, mengahapus, dan mencari lowongan kerja.

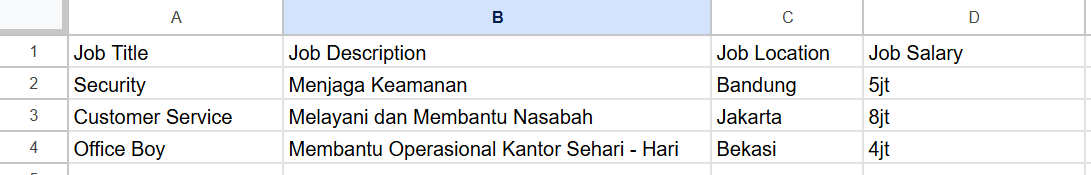
Html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Job Management</title>    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">    <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>  </head>  <body>    <div class="container mt-5">      <h1 class="text-center">Job Management</h1>      <form id="jobForm" class="mb-4">        <div class="mb-3">          <label for="title" class="form-label">Job Title</label>          <input type="text" class="form-control" id="title" required>        </div>        <div class="mb-3">          <label for="description" class="form-label">Job Description</label>          <textarea class="form-control" id="description" required></textarea>        </div>        <div class="mb-3">          <label for="location" class="form-label">Location</label>          <input type="text" class="form-control" id="location" required>        </div>        <div class="mb-3">          <label for="salary" class="form-label">Salary</label>          <input type="text" class="form-control" id="salary" required>        </div>        <input type="hidden" id="jobId">        <button type="button" class="btn btn-primary" onclick="addOrUpdateJob()">Add/Update Job</button>      </form>      <div class="mb-4">        <input type="text" id="searchQuery" class="form-control" placeholder="Search jobs">        <button type="button" class="btn btn-secondary mt-2" onclick="searchJobs()">Search</button>      </div>      <div id="jobList" class="mt-4">        <h2>Job Listings</h2>        <table class="table table-bordered">          <thead>            <tr>              <th>ID</th>              <th>Title</th>              <th>Description</th>              <th>Location</th>              <th>Salary</th>              <th>Actions</th>            </tr>          </thead>          <tbody id="jobs"></tbody>        </table>      </div>    </div>    <script>      const API\_URL = 'https://script.google.com/macros/s/AKfycbwQv9VUznPKvNX-ZnxFVwRiUH1T\_  n8RN1JlZTBK-E6I61FReq5YO-B9QpjYOhX7Wx6d/exec'; // Replace with your Web App URL      function addOrUpdateJob() {        const id = document.getElementById('jobId').value;        const title = document.getElementById('title').value;        const description = document.getElementById('description').value;        const location = document.getElementById('location').value;        const salary = document.getElementById('salary').value;        const action = id ? 'update' : 'add';        const params = id ? `&id=${id}` : '';          axios.get(`${API\_URL}?action=${action}&title=${title}&description=${description}&location=$  {location}&salary=${salary}${params}`)          .then(response => {            alert(response.data.message);            document.getElementById('jobForm').reset();            document.getElementById('jobId').value = '';            loadJobs();          })          .catch(error => console.error(error));      }      function deleteJob(id) {        axios.get(`${API\_URL}?action=delete&id=${id}`)          .then(response => {            alert(response.data.message);            loadJobs();          })          .catch(error => console.error(error));      }      function searchJobs() {        const query = document.getElementById('searchQuery').value;        axios.get(`${API\_URL}?action=search&query=${query}`)          .then(response => {            displayJobs(response.data.results);          })          .catch(error => console.error(error));      }      function loadJobs() {        axios.get(`${API\_URL}?action=search&query=`)          .then(response => {            displayJobs(response.data.results);          })          .catch(error => console.error(error));      }      function displayJobs(jobs) {        const jobList = document.getElementById('jobs');        jobList.innerHTML = '';        jobs.forEach((job, index) => {          const row = document.createElement('tr');          row.innerHTML = `            <td>${index + 1}</td>            <td>${job[0]}</td>            <td>${job[1]}</td>            <td>${job[2]}</td>            <td>${job[3]}</td>            <td>              <button class="btn btn-warning btn-sm" onclick="editJob(${index})">Edit</button>              <button class="btn btn-danger btn-sm" onclick="deleteJob(${index})">Delete</button>            </td>          `;          jobList.appendChild(row);        });      }      function editJob(id) {        axios.get(`${API\_URL}?action=search&query=`)          .then(response => {            const job = response.data.results[id];            document.getElementById('jobId').value = id;            document.getElementById('title').value = job[0];            document.getElementById('description').value = job[1];            document.getElementById('location').value = job[2];            document.getElementById('salary').value = job[3];          })          .catch(error => console.error(error));      }      document.addEventListener('DOMContentLoaded', loadJobs);    </script>  </body>  </html> |

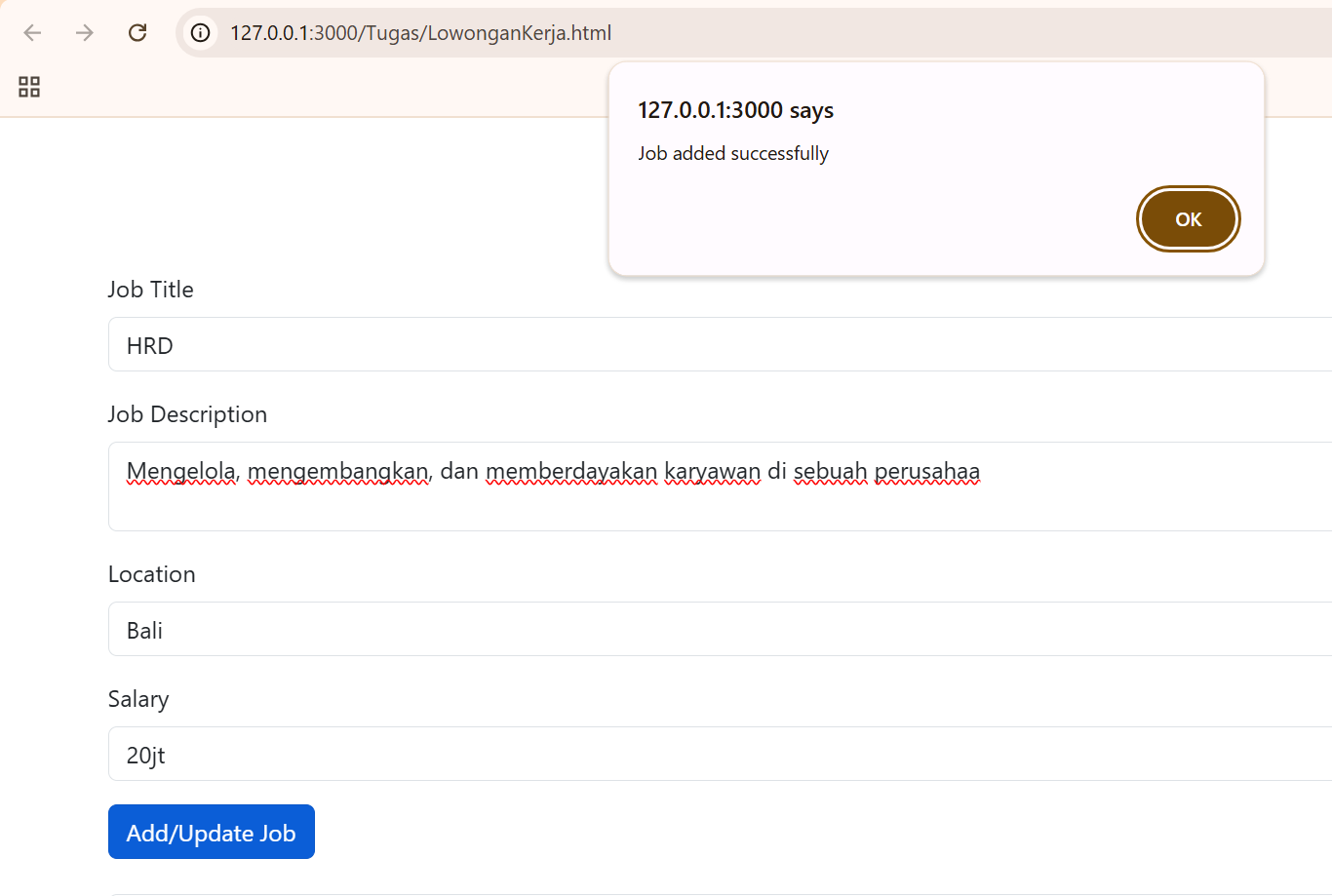
Apps Script

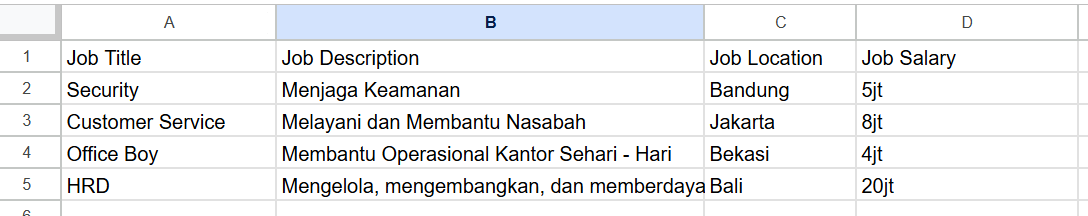
|  |
| --- |
| function doGet(e) {    const action = e.parameter.action;    const sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();      switch (action) {      case 'add':        return addJob(e, sheet);      case 'update':        return updateJob(e, sheet);      case 'delete':        return deleteJob(e, sheet);      case 'search':        return searchJobs(e, sheet);      default:        return ContentService.createTextOutput(JSON.stringify({ error: 'Invalid action' })).setMimeType(ContentService.MimeType.JSON);    }  }  function addJob(e, sheet) {    const { title, description, location, salary } = e.parameter;    if (!title || !description || !location || !salary) {      return ContentService.createTextOutput(JSON.stringify({ error: 'Missing required fields' })).setMimeType(ContentService.MimeType.JSON);    }    sheet.appendRow([title, description, location, salary]);    return ContentService.createTextOutput(JSON.stringify({ message: 'Job added successfully' })).setMimeType(ContentService.MimeType.JSON);  }  function updateJob(e, sheet) {    const { id, title, description, location, salary } = e.parameter;    const row = parseInt(id) + 2; // Adjusting for 1-based index in Sheets, and skipping the header row    if (!title || !description || !location || !salary) {      return ContentService.createTextOutput(JSON.stringify({ error: 'Missing required fields' })).setMimeType(ContentService.MimeType.JSON);    }    sheet.getRange(row, 1).setValue(title);    sheet.getRange(row, 2).setValue(description);    sheet.getRange(row, 3).setValue(location);    sheet.getRange(row, 4).setValue(salary);    return ContentService.createTextOutput(JSON.stringify({ message: 'Job updated successfully' })).setMimeType(ContentService.MimeType.JSON);  }  function deleteJob(e, sheet) {    const { id } = e.parameter;    const row = parseInt(id) + 2; // Adjusting for 1-based index in Sheets, and skipping the header row    sheet.deleteRow(row);    return ContentService.createTextOutput(JSON.stringify({ message: 'Job deleted successfully' })).setMimeType(ContentService.MimeType.JSON);  }  function searchJobs(e, sheet) {    const { query } = e.parameter;    const data = sheet.getDataRange().getValues();    const headers = data.shift(); // Remove the headers    const results = data.filter(row => row.some(cell => cell.toString().toLowerCase().includes(query.toLowerCase())));    return ContentService.createTextOutput(JSON.stringify({ results })).setMimeType(ContentService.MimeType.JSON);  } |

Google Sheets :

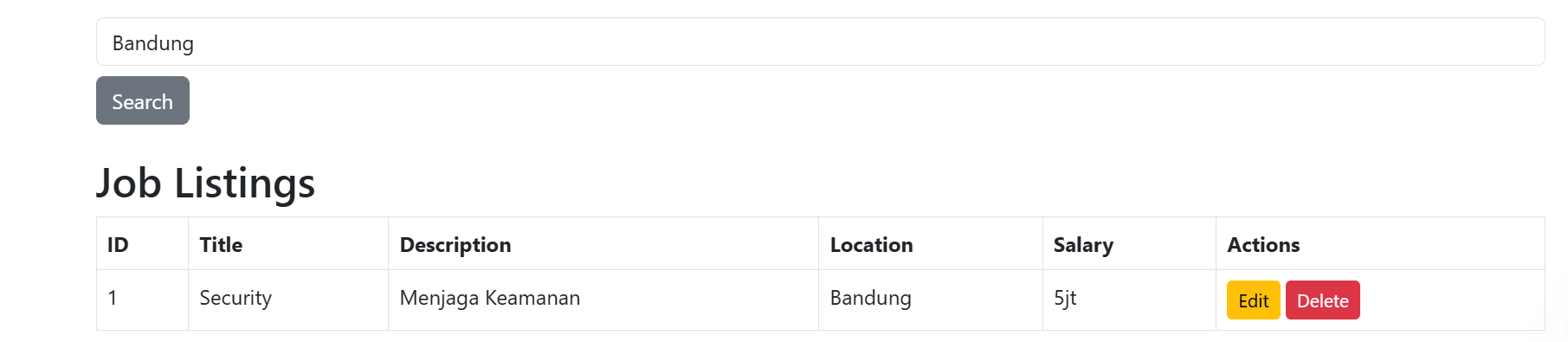


Output :

Add Job :  




Search Job :



Remove Job :  
