BAB 5 IMPLEMENTASI

5.1 HTTP *Server*

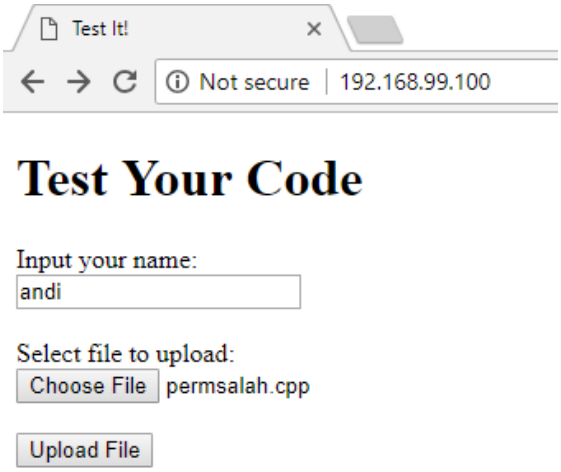
Pada HTTP *server* akan digunakan 3 kode program. Kode program pertama akan digunakan untuk masukan dari pengguna. Pada baris 1-6 digunakan untuk mendefinisikan *title* dari halaman dan juga mengatur halaman web. Lalu baris 7-15 akan digunakan untuk membuat form. Pada form akan terdapat dua masukan dari pengguna yaitu nama dan *file* program yang akan diuji yang terlihat pada baris 10 dan baris 12 sampai 13. Nama akan digunakan untuk membedakan proses pengujian. Untuk masukan akan digunakan form yan satu bertipe text dan satunya lagi adalah *file*. Pada baris 7 akan ditentukan halaman selanjutnya, dimanan saat pengguna menekan tombol *Upload File*, maka inputan akan dikirim menggunakan method post ke kode program selanjutnya untuk menyimpan *file*. *File* akan dicek terlebih dahulu apakah terdapat *file* yang sama, ukurannya cukup dan tipe *file*. Jika semua sudah benar, *file* akan disimpan lalu halaman ini juga akan menyimpan cookie yang akan digunakan untuk kode selanjutnya. Implementasi dapat dilihat pada algoritme 1. Sedangkan tampilan dapat dilihat pada gambar 5.1

Algoritme 1:Implementasi Halaman Utama

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Test It!</title>
5  </head>
6  <body>
7  <h1> Test Your Code</h1><form action="upload.php" method="post"
8  enctype="multipart/form-data">
9           Input your name:<br>
10          <input type="text" name="nama" id="nama"><br>
11     <br>Select file to upload:<br>
12     <input type="file" name="fileToUpload"
13 id="fileToUpload"><br>
14     <br><input type="submit" value="Upload File" name="submit">
15 </form>
16 </body>
17 </html>
```

Pada program upload *file* / menyimpan *file*, algoritmenya dapat dilihat pada algoritme 2. Pada baris 2-6 nama dari *file* akan diambil dan nama *file* nantinya akan diganti dengan nama yang ada dari form sebelumnya. Pada baris ke-7 dideklarasikan variabel baru dengan nilai satu. Variabel baru ini nantinya akan berubah nilainya menjadi nol saat tidak lolos pengecekan. Pada baris 9-11 akan dilakukan pengecekan apakah *file* dengan nama yang sama sudah ada sebelumnya atau belum. Ini dilakukan sehingga program *websocket server* tidak akan bingung dalam mengambil *file*. Pada baris 13-15 akan dilakukan pengecekan apakah *file* yang akan diunggah tidak melebihi 500.000 *bytes.* Pada baris 17-20 dilakukan pengecekan sehingga *file* yang diunggah hanya *file* dengan bahasa pemrograman C++, Java dan Python. Pada baris 22 dan 23 akan dilakukan seleksi apabila variabel yang tadi dibuat bernilai 0, maka *file* tidak akan tersimpan dan akan menampilkan tulisan permohonan maaf. Pada baris 24-31 merupakan proses untuk menyimpan

file pada direktori yang sebelumnya telah ditentukan dan menyimpan nama serta jenis *file* pada *cookies* dan selanjutnya akan membuka halaman hasil. Baris 32 dan 33 digunakan pada saat jika terjadi kesalahan pada saat mengunggah *file* maka halaman akan menampilkan pesan adanya kesalahan.



Gambar 5.1 Halaman awal

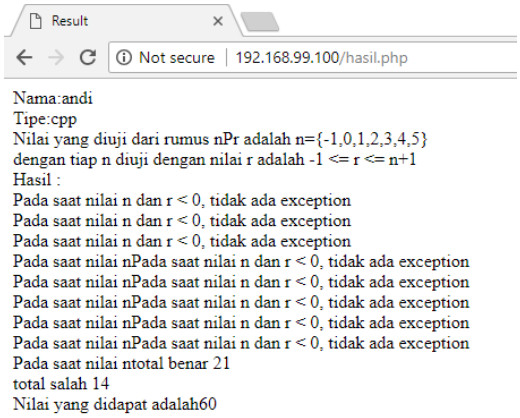
Algoritme 2 : Implementasi menyimpan *file*

```
1   <?php
2   $imageFileType =
3  strtolower(pathinfo($_FILES["fileToUpload"]["name"],PATHINFO_EXTEN
4  SION));
5   $target_dir = "D:\Smt akhir\beneran/";
6   $target_file = $target_dir.$_POST["nama"].".".$imageFileType;
7   $uploadOk = 1;
8
9   if (file_exists($target_file)) {
10      echo "Sorry, file already exists.";
11      $uploadOk = 0;
12  }
13  if ($_FILES["fileToUpload"]["size"] > 500000) {
14      echo "Sorry, your file is too large.";
15      $uploadOk = 0;
16  }
17  if($imageFileType != "py" && $imageFileType != "java" &&
18 $imageFileType != "cpp") {
19      echo "Sorry, ";
20      $uploadOk = 0;
21  }
22  if ($uploadOk == 0) {
23      echo "Sorry, your file was not uploaded.";
24  } else {
25      if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"],
26 $target_file)) {
27          echo "The file ". basename(
28 $_FILES["fileToUpload"]["name"]). " has been uploaded.";
29          setcookie("nama",$_POST['nama']);
30          setcookie("tipe",$imageFileType);
31          header("Location: hasil.php");
32      } else {
33          echo "Sorry, there was an error uploading your file.";
34      }
35  }
36  ?>
```

Pada halaman hasil, pada baris satu dan dua digunakan untuk menentukan tulisan *title*. Selanjutnya antara baris 3-30 merupakan *script* yang akan dijalankan pada saat halaman diakses. Baris 5-8 akan membuka koneksi ke *websocket server* lalu menjalankan fungsi kirim data. Baris 11-13 akan berjalan pada saat ada data masuk, maka fungsi *set* hasil akan berjalan. Baris 15-24 merupakan fungsi kirim data. Fungsi ini akan mengambil data pada *cookies* lalu mengirimkannya ke koneksi yang tadi telah dibuka dalam bentuk *Json*. Baris 26-28 merupakan fungsi untuk menerima data. Pada fungsi ini pada saat terdapat data yang diterima, data tersebut akan ditampilkan pada bagian hasil pada badan halaman. Baris 33 dan 34 akan menampilkan data yang tadi disimpan pada *cookies*. Baris 35-37 merupakan deklarasi dari bagian hasil, dimana pada bagian ini nantinya akan ditampilkan data hasil pengujian kode program yang diterima. Contoh tampilan dapat dilihat pada gambar 5.2. Implementasi dapat dilihat pada algoritme 3.

Algoritme 3: Implementasi Halaman Hasil

```
1     <head>
2         <title>Result</title>
3         <script type="text/javascript">
4
5              ws = new Websocket('ws://server:port/');
6              ws.onopen = function(){
7                  // alert("hello");
8                  kirimData();
9              };
10
11               ws.onmessage = function(evt){
12                     //alert(evt.data);
13                     setHasil(evt.data);
14               };
15             function kirimData(){
16                 cook = document.cookie;
17                 cookiearray = cook.split('; ');
18                 nama = cookiearray[1].split("=")[1];
19                 nama = nama.replace(/[+]/g, ' ');
20                 tipe = cookiearray[0].split("=")[1];
21                 document.getElementById('nama').innerHTML = nama;
22                 document.getElementById('tipe').innerHTML = tipe;
23                 var nudata= {type:tipe,name:nama};
24                 ws.send(JSON.stringify(nudata));
25             }
26               function setHasil(data){
27                     hasil = document.getElementById('hasil');
28                     hasil.innerHTML = data
29               }
30         </script>
31     </head>
32     <body>
33     Nama:<div id="nama" style="display: inline-block"></div><br>
34     Tipe:<div id="tipe" style="display: inline-block"></div><br>
35     Hasil :
36         <div id="hasil" nama="hasil" style="display: contents;">
37         </div>
38 </body>
```



Gambar 5.2 Tampilan Halaman Hasil

5.2 Implementasi Web *Socket Server*

Untuk web *socket server* akan digunakan bahasa python dengan menggunakan beberapa modul. Modul yang digunakan antara lain *twisted*, *txws*, *json*, *os* dan *rpyc*. Pada baris pertama akan dideklarasikan variabel counter untuk *load balancing* dengan nilai awal 0. Lalu pada baris dua akan dibuat fungsi yang akan berjalan saat mendapatkan data dari http *server*. Pada baris tiga dan empat, data dari http *server* akan disimpan pada variabel nama dan bahasa/tipe pengujian. Setelah itu pada baris lima variabel counter akan ditambah satu untuk merepresentasikan bahwa terdapat pengujian baru. Setelah itu pada baris 6-11 nilai dari variabel counter akan dihitung dengan modulo 3 untuk menentukan pada slave yang mana pengujian akan dilakukan yang selanjutnya akan langsung membuka koneksi dengan *slave*. Pada baris dua belas akan membuka *file* sesuai dengan nama dan tipe *file*. Pada baris 13-18 akan menyeleksi sesuai dengan tipe *file* dan akan memanggil fungsi yang sesuai dengan bahasa pemrograman. Setelah itu pada baris 19 hasil yang dikembalikan dari slave akan dikembalikan kepada http *server*. Http *server* nantinya akan menampilkan hasil kepada pengguna. Implementasi dapat dilihat pada algoritme 4.

Algoritme 4: *Websocket Server*

```
1     counter = 0
2      def dataReceived(data):
3             nama = data ["name"]
4             tipe = data ["type"]
5             counter += 1
6             if (counter%3==0):
7                 connect(Slave1)
8             elif(counter%3==1):
9                 connect(Slave2)
10            else:
11                connect(Slave3)
12            file = open(savedfile)
13            if(tipe == "python"):
14                hasil = python(file)
15            elif(tipe == "java"):
16                hasil java(file)
17            elif(tipe == "cpp"):
18                hasil = cpp(file)
19            return hasil
```

5.3 Implementasi *Slave*

Slave akan dibuat menggunakan bahasa python. Pada program ini terdapat satu fungsi untuk setiap Bahasa yang akan diuji. Sebelumnya telah disimpan terlebih dahulu *file-file* yang akan digunakan untuk menjalankan pengujian. Implementasi dari program dapat dilihat pada algoritme 5.

Pada fungsi untuk menguji bahasa python yang terdapat pada baris 1-6, di baris kedua akan dibuat terlebih dahulu dibuat folder untuk menyimpan hasil dan mengumpulkan semua kode sumber yang dibutuhkan. Setelah itu pada baris 3 dan 4 *file* dari *master* dan *file* yang telah disimpan sebelumya akan dicopy dan disimpan pada folder tersebut. Setelah itu pada baris 5 program penguji akan dijalankan pada docker. Hasilnya nanti akan keluar pada folder tersebut. Pada baris 6 hasil tadi akan dikembalikan ke *websocket server*.

Pada fungsi untuk menguji bahasa Java yang dituliskan pada baris 7-13, sebenarnya hampir sama dengan fungsi untuk bahasa python. Pertama pada baris 8 akan dibuat terlebih dahulu dibuat folder untuk menyimpan hasil dan mengumpulkan semua kode sumber yang dibutuhkan. Setelah itu pada baris 9 dan 10 *file* dari *master* dan *file* yang telah disimpan sebelumya akan dicopy dan disimpan pada folder tersebut. Lalu selanjutnya yang membedakan adalah pada baris 11 program harus dilakukan compile terlebih dahulu. Setelah itu pada baris 12 program penguji baru bisa dijalankan. Terakhir pada baris 13 hasil akan dikembalikan juga kepada *websocket server*.

Pada fungsi untuk menguji bahasa C++ yang terdapat pada baris 14-19, di baris 15 akan dibuat terlebih dahulu dibuat folder untuk menyimpan hasil dan mengumpulkan semua kode sumber yang dibutuhkan. Setelah itu pada baris 16 dan 17 *file* dari *master* dan *file* yang telah disimpan sebelumya akan dicopy dan disimpan pada folder tersebut. Setelah itu pada baris 18 program penguji akan dijalankan pada docker. Hasilnya nanti akan keluar pada folder tersebut. Pada baris 19 hasil tadi akan dikembalikan ke *websocket server*.

Algoritme 5: Slave

```
1     def python(file):
2           system(create new folder)
3           system(copy tester program to folder)
4           system(copy file to folder)
5           system(docker run tester)
6           return hasil
7       def java(file):
8           system(create new folder)
9           system(copy tester program to folder)
10          system(copy file to folder)
11          system(docker compile tester)
12          system(docker run tester)
13          return hasil
14      def cpp(file):
15          system(create new folder)
16          system(copy tester program to folder)
17          system(copy file to folder)
18          system(docker run tester)
19          return hasil
```

Pada docker di tiap *slave*, akan dipasang tiga image yang berbeda menggunakan perintah pull. Image yang dipasang adalah python versi 2-slim dengan perintah pada baris 1, gcc versi 7 dengan perintah pada baris 2 dan openjdk versi 11-slim dengan perintah pada baris 3. Perintah lengkap dapat dilihat pada algoritme 6.

Kode Sumber 5.1 Memasang Image pada Docker

```
Algoritme 6: Instalasi Docker Image
1  docker pull python:2-slim
2  docker pull gcc:7
3  docker pull openjdk:11-slim
```