BAB VI
PENUTUP

6.1 Kesimpulan

Berdasarkan perancangan dan pengujian *Application Programming Interface Server* untuk Arduino serta data-data yang telah diperoleh dan dianalisa, maka dapat disimpulkan sebagai berikut :

1. *Application Programming Interface Server* untuk Arduino adalah suatu sistem yang memungkinkan sebuah aplikasi yang berjalan pada jaringan komputer dapat berinteraksi dengan arduino melalui *Application Programming Interface Server.* Prinsip kerja *Application Programming Interface Server* untuk arduino adalah menerima dan mengolah HTTP request dan menterjemahkan HTTP request menjadi TCP *stream* yang akan dikirimkan ke arduino. Arduino akan menterjemahkan TCP *stream* dari server menjadi fungsi-fungsi manipulasi input dan output. Apabila fungsi tersebut telah berhasil dieksekusi, arduino akan mengirimkan TCP *stream* ke server sebagai balasan. Server akan mengirimkan HTTP response dan data JSON yang sesuai dengan request dari client.
2. *Application Programming Interface Server* untuk Arduino hasil dari perancangan diimplementasikan pada server dengan menggunakan bahasa pemrograman skrip PHP dan *Arduino Programming Language* pada Arduino. Server dapat menangani operasi input dan output pada arduino yaitu *digitalRead, analogRead, digitalWrite* dan *analogWrite*.
3. Dari hasil pengujian dengan menggunakan satu buah request, d*elay* total rata-rata yang didapat adalah 14,8 ms yang masih dalam batas wajar dan tidak akan terasa pengaruhnya oleh client sedangkan pada pengujian dengan mengantrikan lebih dari empat buah request pada server terjadi penurunan performa sistem yang ditandai dengan nilai *delay* total rata-rata yang naik pada request kelima dan seterusnya.

6.2 Saran

1. Membuat mekanisme *caching*, sehingga dapat membatasi request yang berulang-ulang pada arduino.
2. Menambahkan metode lain untuk dapat menggunakan *library* lainnya seperti *Servo* atau LCD, sehingga fungsi dari API Server menjadi semakin banyak.
3. Membuat prosedur autentikasi yang lebih baik lagi.

DAFTAR PUSTAKA

Anonim 2013. " Arduino Ethernet Shield ". Arduino. http://arduino.cc/en/Main/ArduinoEthernetShield

Anonim 2013. "Arduino Reference". Arduino. http://arduino.cc/en/Reference/HomePage

Mansfield, Niall. 2003. "*Practical TCP/IP – Mendesain, Menggunakan dan Troubleshooting Jaringan.*" Jogjakarta: Andi

Masse, Mark. 2011. "*REST API Design Rulebook – Designing Consistent RESTful Web Service Interfaces.*" O'Reilly Media

Purbo, Onno W. 2001. "*TCP/IP – Standard, Desain, dan Implementasi.*" Jakarata: PT. Elex Media Komputindo

RFC 2616. 1999. "*Hypertext Transfer Protocol*". R. Fielding.

RFC 793. 1981. "*Transmission Control Protocol*". J. Postel.

Richardson, Leonard dan Sam Ruby. 2007. "*RESTful Web Services – Web Services for the real world.*" O'Reilly Media

LAMPIRAN

```php
<?php
/**
 * Class Name       : ArduinoServer
 * Author           : Samuel Sena
 * Description      : REST API Server for Arduino. Access Arduino I/O  Function via
                      HTTP request. Use JSON as Data Interchange Format
 * 1.0.0  - 25 Feb 2013
 * 1.0.1  - 3 May 2013
 *
 */

class ArduinoServer {

    /**
     * Constant
     */
    const HTTP_GET              = 'GET';
    const HTTP_POST             = 'POST';
    const API_KEY               = '75f5750f6dd6afbec57b0928a0ec306b';
    const INFO_COMMAND          = 'info';
    const READ_COMMAND          = 'read';
    const WRITE_COMMAND         = 'write';
    const READ_MESSAGE          = '1';
    const WRITE_MESSAGE         = '2';
    const DIGITAL_MESSAGE       = '1';
    const ANALOG_MESSAGE        = '2';

    /**
     * Properties
     */
    private $requestURI;
    private $serverMessage;
    private $arduinoMessage;
    private $verb;
    private $arduinoIP;
    private $arduinoPort;
    private $errorMessage;
    private $maxDigitalPin          = 13;
    private $maxAnalogPin           = 5;
    private $forbiddenDigitalPin    = array('0','1','2','10','11','12','13');
    private $forbiddenAnalogPin;
    private $report =  array(
                           'success'           => 'success',
                           'connerror'         => 'Cannot Connect To Arduino',
                           'urierror'          => 'Bad Request',
                           'modeerror'         => 'Bad Pin Type (Digital - Analog)',
                           'pinerror'          => 'Bad Pin Number',
                           'valueerror'        => 'Bad Value',
                           'permission'        => 'Bad API Key'
                           );
    private $JSON;


    /**
     * Constructor
     */
    public function __construct($config, $debug = false){
        if(!$debug){
                error_reporting(0);
        }
        $this->setArduinoIP($config['arduinoIP']);
        $this->setArduinoPort($config['arduinoPort']);
    }
```

```
/**
 * Setter
 */
public function setArduinoIP($arduinoIP = ''){
    $this->arduinoIP = $arduinoIP;
}

public function setArduinoPort($arduinoPort = ''){
    $this->arduinoPort = $arduinoPort;
}

public function setMaxDigitalPin($maxDigitalPin = ''){
    $this->maxDigitalPin = $maxDigitalPin;
}

public function setMaxAnalogPin($maxAnalogPin = ''){
    $this->maxAnalogPin = $maxAnalogPin;
}

public function setforbiddenDigitalPin($forbiddenDigitalPin = array()){
    $this->forbiddenDigitalPin = $forbiddenDigitalPin;
}

public function setforbiddenAnalogPin($forbiddenAnalogPin = array()){
    $this->forbiddenAnalogPin = $forbiddenAnalogPin;
}


/**
 * Main Function
 */
public function startServer(){
    /* Set HTTP Method and Request-URI */
    $this->verb = (isset($_SERVER['REQUEST_METHOD']) ?
$_SERVER['REQUEST_METHOD'] : '');
    $reqstr = (isset($_SERVER['PATH_INFO']) ? $_SERVER['PATH_INFO'] : '');

    /* Remove first / character on Request-URI */
    $req = substr($reqstr, 1);

    /* Explode Request-URI to array(seqment) */
    $this->requestURI = explode('/', $req);

    $this->handleRequest();
}

public function handleRequest(){
    switch($this->verb){
            case self::HTTP_GET :
                    $this->handleGetRequest($this->requestURI);
                    break;
            case self::HTTP_POST :
                    $this->handlePostRequest($this->requestURI, $_POST);
                    break;
            default :
                    $this->errorMessage = $this->report['urierror'];
                    $this->handleErrorRequest();
                    break;

    }
}

public function handleGetRequest($seqment){
    switch($seqment[0]){
            case self::READ_COMMAND :
                    $this->readPin($seqment);
                    break;
            case self::INFO_COMMAND :
                    $this->getInfo();
                    break;
            default :
                    $this->errorMessage = $this->report['urierror'];
                    $this->handleErrorRequest();
                    break;
    }
}

public function handlePostRequest($seqment, $entity){
    switch($seqment[0]){
            case self::WRITE_COMMAND :
                    $this->WritePin($seqment, $entity);
                    break;
            default :
                    $this->errorMessage = $this->report['urierror'];
                    $this->handleErrorRequest();
                    break;
    }
}

public function handleErrorRequest(){
    $data = array(
                    'status' =>
                            array(
                                    'result'        => 'failed',
                                    'errormsg'      => $this->errorMessage
                            )
            );
    $this->buildJson($data);
    $this->generateHTTPResponse('404');
}

public function readPin($seqment){
    $smsg = $this->validateReadRequest($seqment);
    if($smsg){
            $this->serverMessage = $smsg;
            $amsg = $this->callArduino();
            if($amsg != ''){
                    $this->arduinoMessage = $amsg;
                    $data = $this->buildData(self::READ_COMMAND, $seqment);
                    $this->buildJSON($data);
                    $this->generateHTTPResponse('200');
            }else{
                    $this->errorMessage = $this->report['connerror'];
                    $this->handleErrorRequest();
            }
    }else{
    $this->handleErrorRequest();
    }
}

public function getInfo(){
    $data = $this->buildData(self::INFO_COMMAND);
    $this->buildJSON($data);
    $this->generateHTTPResponse('200');
}

public function writePin($seqment, $entity){
    $smsg = $this->validateWriteRequest($seqment, $entity);
    if($smsg){
            $this->serverMessage = $smsg;
            $amsg = $this->callArduino();
            if($amsg != ''){
                    $this->arduinoMessage = $amsg;
                    $data = $this->buildData(self::WRITE_COMMAND, $entity);
                    $this->buildJSON($data);
                    $this->generateHTTPResponse('200');
            }else{
                    $this->errorMessage = $this->report['connerror'];
                    $this->handleErrorRequest();
            }
    }else{
    $this->handleErrorRequest();
    }
}
```

```
public function validateReadRequest($seqment){
    /* Error flag and message initial value */
    $error = false;
    $message = '';
    if($seqment[0] = self::READ_COMMAND){
        /* Command - 1st Request-URI seqment */
        $message .= self::READ_MESSAGE;
        /* Mode/Pin Type - 2nd Request-URI seqment */
        switch($seqment[1]){
            case 'digital' :
                $message .= self::DIGITAL_MESSAGE;
                /* Pin Number - 3rd Request-URI seqment */
                if(!in_array($seqment[2], $this->forbiddenDigitalPin)){
                    $pin = $seqment[2];
                    if(is_numeric($pin)){
                        if(((int)$pin >= 0) && ((int)$pin <= $this->maxDigitalPin)){
                            $message .= str_pad($pin, 2, '0', STR_PAD_LEFT);
                        }else{
                            $this->errorMessage = $this->report['pinerror'];
                            $error = true;
                            return false;
                        }
                    }else{
                        $this->errorMessage = $this->report['pinerror'];
                        $error = true;
                        return false;
                    }
                }else{
                    $this->errorMessage = $this->report['pinerror'];
                    $error = true;
                    return false;
                }
                break;
            case 'analog' :
                $message .= self::ANALOG_MESSAGE;
                /* Pin Number - 3rd Request-URI seqment */
                if(!in_array($seqment[2], $this->forbiddenAnalogPin)){
                    $pin = $seqment[2];
                    if(is_numeric($pin)){
                        if(((int)$pin >= 0) && ((int)$pin <= $this->maxAnalogPin)){
                            $message .= str_pad($pin, 2, '0', STR_PAD_LEFT);
                        }else{
                            $this->errorMessage = $this->report['pinerror'];
                            $error = true;
                            return false;
                        }
                    }else{
                        $this->errorMessage = $this->report['pinerror'];
                        $error = true;
                        return false;
                    }
                }else{
                    $this->errorMessage = $this->report['pinerror'];
                    $error = true;
                    return false;
                }
                break;
            default:
                $this->errorMessage = $this->report['modeerror'];
                $error = true;
                break;
        }

        if(!$error){
            $message .= "\n"; //End Message
            return $message;
        }else{
            return false;
        }

    }else{
        return false;
    }
}
```

```
public function validateWriteRequest($segment, $entity){
    /* Error flag and message initial value */
    $error = false;
    $message = '';
    if($segment[0] = self::WRITE_COMMAND){
        if($entity['apiKey'] == self::API_KEY){
            /* Command - 1st Request-URI segment */
            $message .= self::WRITE_MESSAGE;
            /* Mode/Pin Type - Mode Post Entity */
            switch($entity['mode']){
                case 'digital' :
                  $message .= self::DIGITAL_MESSAGE;
                  /* Pin Number - Pin Post Entity */
                  if(!in_array($entity['pin'], $this->forbiddenDigitalPin)){
                    $pin = $entity['pin'];
                    if(is_numeric($pin)){
                        if(((int)$pin >= 0) && ((int)$pin <= $this->maxDigitalPin)){
                            $message .= str_pad($pin, 2, '0', STR_PAD_LEFT);
                        }else{
                            $this->errorMessage = $this->report['pinerror'];
                            $error = true;
                            return false;
                        }
                    }else{
                        $this->errorMessage = $this->report['pinerror'];
                        $error = true;
                        return false;
                    }
                  }else{
                    $this->errorMessage = $this->report['pinerror'];
                    $error = true;
                    return false;
                  }
                  /* Value (0-1) - 4th Value Post Entity */
                  $value = $entity['value'];
                  if(is_numeric($value)){
                    if(($value >= 0) && ($value <= 1)){
                        $message .= str_pad($value, 3, '0', STR_PAD_LEFT);
                    }else{
                        $this->errorMessage = $this->report['valueerror'];
                        $error = true;
                        return false;
                    }
                  }else{
                    $this->errorMessage = $this->report['valueerror'];
                    $error = true;
                    return false;
                  }
                  break;

                  case 'analog' :
                    $message .= self::ANALOG_MESSAGE;
                    /* Pin Number - 3rd Request-URI segment */
                    if(!in_array($entity['pin'], $this->forbiddenDigitalPin)){
                      $pin = $entity['pin'];
                      if(is_numeric($pin)){
                        if(((int)$pin >= 0) && ((int)$pin <= $this->maxDigitalPin)){
                            $message .= str_pad($pin, 2, '0', STR_PAD_LEFT);
                        }else{
                            $this->errorMessage = $this->report['pinerror'];
                            $error = true;
                            return false;
                        }
                      }else{
                        $this->errorMessage = $this->report['pinerror'];
                        $error = true;
                        return false;
                      }
                    }else{
                      $this->errorMessage = $this->report['pinerror'];
                      $error = true;
                      return false;
                    }
                    /* PWM Value (0-255) - 4th Value Post Entity */
                    $value = (int)$entity['value'];
                        if(is_numeric($value)){
                            if(($value >= 0) && ($value <= 255)){
                                $message .= str_pad($value, 3, '0', STR_PAD_LEFT);
                            }else{
                                $this->errorMessage = $this->report['valueerror'];
                                $error = true;
                                return false;
                            }
                        }else{
                            $this->errorMessage = $this->report['valueerror'];
                            $error = true;
                            return false;
                        }

                        break;
                    default:
                        $this->errorMessage = $this->report['modeerror'];
                        $error = true;
                        break;
                }

                if(!$error){
                    $message .= "\n"; //End Message
                    return $message;
                }else{
                    return false;
                }

            }else{
                $this->errorMessage = $this->report['permission'];
                return false;
            }

        }else{
            $this->errorMessage = $this->report['urierror'];
            return false;
        }
}


public function callArduino(){
        $socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
        $message = $this->serverMessage;
        //Set Timeout, Connect and Send Message
        if(socket_connect($socket, $this->arduinoIP, $this->arduinoPort)){
            if(socket_write($socket, $message, strlen($message))){
                $response = socket_read($socket, 1024, PHP_BINARY_READ);
                if($response != ''){
                    socket_close($socket);
                    return $response;
                }else{
                    $this->errorMessage = $this->report['connerror'];
                    socket_close($socket);
                    return false;
                }
            }
        socket_close($socket);
        }else{
            $this->errorMessage = $this->report['connerror'];
            return false;
        }
    }

    public function buildData($command, $params){
        switch($command){
            case self::READ_COMMAND :
                $data = array(
                            'status' =>
                                array(
                                    'action' => 'read',
                                    'result' => $this->report['success']
                                ),
                            'data' =>
                                array(
                                    'mode'      => $params[1],
                                    'pin'       => $params[2],
                                    'value'     => (int)$this->arduinoMessage
                                )
                        );
                break;

            case self::WRITE_COMMAND :
                $data = array(
                            'status' =>
                                array(
                                    'action' => 'write',
                                    'result' => $this->report['success']
                                ),
                            'data' =>
                                array(
                                    'mode'      => $params['mode'],
                                    'pin'       => $params['pin'],
                                    'value'     => (int)$this->arduinoMessage
                                )
                        );
                break;

            case self::INFO_COMMAND :
                $data = array(
                    'arduinoIP'     => $this->arduinoIP,
                    'arduinoPort'   => $this->arduinoPort,
                    'maxDigitalPin' => $this->maxDigitalPin,
                    'maxAnalogPin' => $this->maxAnalogPin,
                    'forbiddenPin' => $this->forbiddenPin
                );
                break;

            default :
                break;
        }
        return $data;
    }

    public function buildJSON($data){
        $this->JSON = json_encode($data, JSON_PRETTY_PRINT);
    }

    public function generateHTTPResponse($code){
        if($code == '200'){
            header('HTTP/1.1 200 OK');
        }else{
            header('HTTP/1.1 404 Not Found');
        }

        header('Content-type: application/json');
        echo $this->JSON;
    }
}
?>
```

```php
<?php
/**
 * Filename    : digwrite.php
 * Author      : Samuel Sena
 * Description : PHP File to send HTTP Request to execute digitalWrite function
 *
 */

$url = 'http://172.16.1.1/write';
$fields = array(
                       'apiKey'        => '75f5750f6dd6afbec57b0928a0ec306b',
                       'mode'          => 'digital',
                       'pin'           => '3',
                       'value'         => 1,
               );

$fields_string = http_build_query($fields);

$ch = curl_init();
curl_setopt($ch,CURLOPT_URL, $url);
curl_setopt($ch,CURLOPT_POST, count($fields));
curl_setopt($ch,CURLOPT_POSTFIELDS, $fields_string);
$result = curl_exec($ch);
$info = curl_getinfo($ch);
curl_close($ch);
?>
```

```php
<?php
/**
 * Filename    : analogwrite.php
 * Author      : Samuel Sena
 * Description : PHP File to send HTTP Request to execute analogWrite function
 *
 */

$url = 'http://172.16.1.1/write';
$fields = array(
                       'apiKey'        => '75f5750f6dd6afbec57b0928a0ec306b',
                       'mode'          => 'analog',
                       'pin'           => '3',
                       'value'         => 63,
               );

$fields_string = http_build_query($fields);

$ch = curl_init();
curl_setopt($ch,CURLOPT_URL, $url);
curl_setopt($ch,CURLOPT_POST, count($fields));
curl_setopt($ch,CURLOPT_POSTFIELDS, $fields_string);
$result = curl_exec($ch);
$info = curl_getinfo($ch);
curl_close($ch);
?>
```

```
<?php
/**
 * Filename    : delay.php
 * Author      : Samuel Sena
 * Description : PHP File to send 6 Paralel HTTP Request to test Delay
 */

$ch = array();

for($i=0;$i<=5;$i++){
        $ch[$i] = curl_init();
        curl_setopt($ch[$i], CURLOPT_URL, "http://172.16.1.1/read/digital/3");
        curl_setopt($ch[$i], CURLOPT_HEADER, 0);
}

$mh = curl_multi_init();

for($i=0;$i<=5;$i++){
        curl_multi_add_handle($mh,$ch[$i]);
}

$active = null;

do{
        $mrc = curl_multi_exec($mh, $active);
} while ($mrc == CURLM_CALL_MULTI_PERFORM);

while ($active && $mrc == CURLM_OK) {
    if (curl_multi_select($mh) != -1) {
        do {
            $mrc = curl_multi_exec($mh, $active);
        } while ($mrc == CURLM_CALL_MULTI_PERFORM);
    }
}

for($i=0;$i<=5;$i++){
        curl_multi_remove_handle($mh, $ch[$i]);
}

curl_multi_close($mh);
?>
```

```
/**
 * Filename    : arduino.ino
 * Author      : Samuel Sena
 * Description : Arduino Program to handle TCP Stream
 */

#include <SPI.h>
#include <Ethernet.h>

int index = 0;
char messageBuffer[13];
char cmd[3];
char pin[3];
char val[4];
char response[5];

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress arduinoIP(192,168,1,2);
int port = 3000;

EthernetServer server(port);

void setup() {
  Serial.begin(9600);
  Ethernet.begin(mac, arduinoIP);
  server.begin();
  Serial.print("Arduino is at ");
  Serial.println(Ethernet.localIP());
}

void loop() {
  EthernetClient socketClient = server.available();
  if(socketClient){
    Serial.println("Server Connected");
    while(socketClient.connected()){
      if(socketClient.available()) {
          char x = socketClient.read();
          if (x == '\n'){
            process();
            socketClient.print(response);
            Serial.println(response);
          }else{
            messageBuffer[index++] = x;
          }
      }
    }
    Serial.println("Server Disconnected");
    socketClient.stop();
  }
}

/*
 * Parse TCP Stream
 */
void process() {
  index = 0;

  Serial.println(messageBuffer);
  strncpy(cmd, messageBuffer, 2);

  strncpy(pin, messageBuffer + 2, 2);

  strncpy(val, messageBuffer + 4, 3);

  int cmdid = atoi(cmd);

  switch(cmdid) {
    case 11:  digRead(pin);             break;
    case 12:  anRead(pin);              break;
    case 21:  digWrite(pin,val);        break;
    case 22:  anWrite(pin,val);         break;
    default:                            break;
  }
}
```

```
/*
 * Set pin mode
 */
void pMode(char *pin, char *val) {
  int p = getPin(pin);
  if(p == -1){
    return;
  }
  if (atoi(val) == 0) {
    pinMode(p, OUTPUT);
  } else {
    pinMode(p, INPUT);
  }
}

/*
 * Digital write
 */
void digWrite(char *pin, char *val) {
  int p = getPin(pin);
  if(p == -1) {
    return;
  }
  pinMode(p, OUTPUT);
  char message[3];
  if (atoi(val) == 0) {
    digitalWrite(p, LOW);
    sprintf(message, "%1d", 0);
    generateSuccessResponse(message);
  } else {
    digitalWrite(p, HIGH);
    sprintf(message, "%1d", 1);
    generateSuccessResponse(message);
  }
}

/*
 * Digital read
 */
void digRead(char *pin) {
  int p = getPin(pin);
  if(p == -1) {
    return;
  }
  pinMode(p, INPUT);
  int value = digitalRead(p);
  char message[4];
  sprintf(message, "%1d", value);
  generateSuccessResponse(message);
}

/*
 * Analog read
 */
void anRead(char *pin) {
  //Serial.println("Analog Read");
  int p = getPin(pin);
  if(p == -1) {
    return;
  }
  pinMode(p, INPUT);
  int value = analogRead(p);
  char message[5];
  sprintf(message, "%4d", value);
  generateSuccessResponse(message);
}

void anWrite(char *pin, char *val) {
  int p = getPin(pin);
  if(p == -1) {
    return;
  }
  pinMode(p, OUTPUT);
  analogWrite(p,atoi(val));
  int value = atoi(val);
  char message[4];
  sprintf(message, "%3d", value);
  generateSuccessResponse(message);
}

int getPin(char *pin) {
  int ret = -1;
    ret = atoi(pin);
    if(ret == 0 && (pin[0] != '0' || pin[1] != '0')) {
      ret = -1;
    }
  return ret;
}

/*
 * Send Stream to Server
 */
void generateSuccessResponse(char *message){
  strncpy(response, message, sizeof(message)+2);
}
```