

# PERTEMUAN 8

## *Web Server GO, JSON, dan Polymer*

### **Menggunakan Bahasa Pemrograman Go**

Objectif :

1. Mahasiswa dapat memahami konsep penggunaan Web Server
2. Mahasiswa dapat memahami pembuatan file JSON dengan golang.
3. Mahasiswa dapat memahami penggunaan *Web Component* pada polymer. .

## 8.1 Arsitektur Web Server Go

Bahasa pemrograman Go menyediakan *package* `net/http` yang berisi berbagai macam fitur untuk keperluan pembuatan aplikasi berbasis web. Termasuk di dalamnya routing, server, templating, dan lainnya. Go memiliki web server sendiri, dan web server tersebut berada di dalam Go, tidak seperti bahasa lain yang servernya terpisah dan perlu di-instal sendiri (seperti PHP yang memerlukan Apache, .NET yang memerlukan IIS). Web Server Go sudah tersedia ketika melakukan instalasi Go pada suatu sistem operasi. Berikut ini adalah cara untuk menjalankan web server golang. :

A screenshot of a code editor with a dark background and light-colored text. The code is a Go program for a web server. It starts with a package declaration, followed by imports for 'fmt' and 'net/http'. The main function calls 'http.HandleFunc' to set a handler for the root path, which prints 'Golang!' to the response writer. Then, it calls 'http.ListenAndServe' to start the server on port 8085. The code is numbered from 1 to 17.

```
1 package main
2
3 import (
4     "fmt"
5     "net/http"
6 )
7
8 func main(){
9
10     http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request){
11         fmt.Fprintln(w, "Golang!")
12     })
13
14     fmt.Println("Menjalankan web server golang dengan port:8085")
15     http.ListenAndServe(":8085", nil)
16 }
17
```

Gambar 8.1 Web Server Go

Penjelasan Kode :

- **Import (“net/http”)**

*Package* “net/http” menyediakan berbagai fitur untuk keperluan pembuatan Aplikasi berbasis web.

- **http.HandleFunc("/", func(w http.ResponseWriter, r \*http.Request) {})**

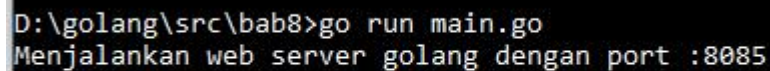
\Merupakan suatu fungsi yang harus tersedia ketika menjalankan web server Go, pada bagian “/” digunakan untuk melakukan modifikasi url ketika menjalan web server Go, jika tanda / diubah menjadi “/belajar” maka url yang semula berupa `http://localhost:8085` akan berubah menjadi `http://localhost:8085/ belajar`.

- **http.ListenAndServe(":8085", nil)**

Digunakan untuk menjalankan web server Go beserta template yang sudah disiapkan sebelumnya, aplikasi Go akan berjalan dengan url <http://localhost:8085>.

## 8.2 Menjalankan Web Server Golang

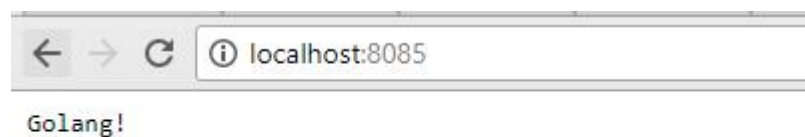
Kekuatan *super power* yang dimiliki Go salah satunya adalah kemudahan dalam membuat dan menjalankan web servernya. Tidak ada cara khusus untuk menjalankan web server Go, cukup menjalankan file main Go melalui command prompt maka web server tersebut akan secara otomatis ikut berjalan, berikut contoh cara menjalankan dan hasil Output -nya :



```
D:\golang\src\bab8>go run main.go
Menjalankan web server golang dengan port :8085
```

Gambar 8.2 Command Prompt

Lalu buka web browser dan ketikkan pada url “localhost:8085” sesuai dengan port pada kode yang dibuat, tampilan web dapat dilihat pada Gambar 8.3.



Gambar 8.3 Tampilan Web

## 8.3 Pembuatan Route dalam Golang

Pada bagian sebelumnya sudah dijelaskan cara menjalankan Aplikasi golang dengan route yang digunakan. Pada sub bab ini akan dibahas lebih men-

detail mengenai cara menggunakan route berdasarkan pengembalian respon dalam web server golang, yaitu sebagai berikut :

1. Respon Data Dari Fungsi Route Secara Langsung

Kode response yang ingin dikembalikan di tuliskan pada fungsi route yang dideklarasikan secara langsung, contoh dari cara ini dapat dilihat pada Gambar 8.1.

2. Respon Data Menggunakan Fungsi Terpisah

Penggunaan cara ini dengan menuliskan kode *response* pada fungsi yang berbeda dengan fungsi route, cara ini dapat menghasilkan kode yang lebih rapih dan terstruktur, penggunaan cara ini dapat dilihat pada Gambar 8.4.



```
1 package main
2
3 import (
4     "fmt"
5     "net/http"
6     "html/template"
7 )
8
9 func main(){
10
11     http.HandleFunc("/fungsi", index)
12
13     fmt.Println("Menjalankan web server golang dengan port :8085")
14     http.ListenAndServe(":8085", nil)
15 }
16
17 func index(w http.ResponseWriter, r *http.Request){
18     fmt.Fprintln(w, "apa kabar!")
19 }
20
```

Gambar 8.4 Respon fungsi terpisah

3. Respon Menggunakan Template Html

Template memberikan kemudahan dalam mendesain tampilan view aplikasi website. Pada Go menyediakan engine template sendiri, dengan banyak fitur yang tersedia di dalamnya. Pada bagian ini akan dibahas contoh sederhana penggunaan template untuk menampilkan data. Hal pertama yang perlu disiapkan dengan membuat template. Buatlah file **index.html** lalu isi dengan kode seperti gambar 8.5 berikut.

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title></title>
5 </head>
6 <body>
7   <p>Halo, selamat belajar {{.nama }}</p>
8 </body>
9 </html>

```

Gambar 8.5 Kode dari index.html

Kode `{{.nama }}` adalah representasi variabel **nama** yang dikirim dari route. Kode tersebut nantinya digantikan dengan isi variabel **nama**. Selanjutnya ubah isi file **main.go** dengan kode berikut

```

1 package main
2
3 import (
4   "fmt"
5   "net/http"
6   "html/template"
7 )
8
9 func main(){
10
11   http.HandleFunc("/template", func(w http.ResponseWriter, r *http.Request){
12     var data = map[string]string{
13       "nama" : "Lepkom",
14     }
15
16     var t, err = template.ParseFiles("index.html")
17     if err != nil {
18       fmt.Println(err.Error())
19       return
20     }
21
22     t.Execute(w, data)
23   })
24
25   fmt.Println("Menjalankan web server golang dengan port :8085")
26   http.ListenAndServe(":8085", nil)
27 }
28

```

Gambar 8.6 Main.go

Kemudian jalankan server, lalu buka <http://localhost:8081/> pada web browser, maka data **nama** akan tampil seperti terlihat pada gambar 8.6.



Gambar 8.7 Tampilan Web

Fungsi **template. ParseFiles( )** digunakan untuk parsing template, mengembalikan 2 data yaitu instance template-nya dan error (jika ada). Pemanggilan method **Execute( )** akan membuat hasil parsing template ditampilkan ke layar web browser.

#### 8.4 JavaScript *Object Notation (JSON)*

*JSON* memiliki singkatan dari JavaScript *Object Notation* yang dimana merupakan sebuah format untuk berbagi data. Seperti dapat kita lihat dari namanya, *JSON* diturunkan dari bahasa pemrograman JavaScript, akan tetapi format ini tersedia bagi banyak bahasa lain termasuk Golang. *JSON* biasanya dilafalkan seperti nama "Jason.". *JSON* menggunakan ekstensi *.JSON* saat ia berdiri sendiri. Saat didefinisikan di dalam format file lain (seperti di dalam *.html*), ia dapat tampil didalam tanda petik sebagai *JSON* string, atau ia dapat dimasukkan kedalam sebuah variabel. Format ini sangat mudah untuk ditransfer antar server web dengan klien atau browser.

Golang menyediakan *package encoding/JSON* yang berisikan banyak fungsi untuk kebutuhan operasi *JSON*. Di bagian ini, akan dibahas bagaimana menggunakan operasi Encode & decode suatu *JSON/object*.

##### 8.4.1 Encode Suatu *Object* Menjadi *JSON*

Encode merupakan pengirim mengkodean informasi yang akan disampaikan ke dalam simbol atau isyarat, pada bagian ini akan membahas cara melakukan encode suatu *object* berupa struct pada golang menjadi *JSON*, sebelum menggunakan fungsi *JSON* pada golang diwajibkan terlebih dahulu

untuk menambahkan *package* tambahan pada *import*, *package* yang digunakan adalah **"encoding/JSON"**, setelah itu fungsi yang digunakan untuk merubah suatu *object* menjadi *JSON* adalah fungsi **json.Marshal**, fungsi ini tidak hanya dapat merubah suatu variabel *object* berupa struct namun dapat juga merubah suatu `map[string]interface{}` atau slice. Berikut adalah contoh merubah suatu data berupa slice dan struct menjadi suatu *JSON*, dapat dilihat pada Gambar 8.8.

```
1 package main
2
3 import (
4     "fmt"
5     "encoding/json"
6 )
7
8 func main(){
9
10    var data_mahasiswa = []lepkom {
11        {
12            Nama : "Aria",
13            Kursus : "Funda Web",
14        },
15    }
16
17    result, err := json.Marshal(data_mahasiswa)
18
19    if err != nil {
20        fmt.Println(err.Error())
21        return
22    }
23
24    var jsonString = string(result)
25    fmt.Println(jsonString)
26
27 }
28
29 type lepkom struct {
30     Nama string `json:"nama_mahasiswa"`
31     Kursus string `json:"kursus_mahasiswa"`
32 }
33
```

Gambar 8.8 Encode Object to JSON

Pertama deklarasikan terlebih dahulu suatu struct mahasiswa dengan 2 *properti* yaitu Nama dan Kursus, kedua *properti* tersebut bertipe data string dengan tambahan nama *properti* **"nama\_mahasiswa"** dan **"kursus\_mahasiswa"** ketika dilakukan suatu operasi *JSON*. Kemudian buat dan isikan suatu variable `data_mahasiswa` bertipe data struct mahasiswa, untuk merubah `data_mahasiswa` menjadi suatu *JSON*, digunakan fungsi `JSON.Marshal(data_mahasiswa)` yang di masukkan kedalam tipe data `result`, lalu untuk melihat apakah data tersebut sudah

berhasil diubah menjadi format data *JSON* dapat dilakukan cetak variabel *result*, untuk Output dari program diatas dapat dilihat pada Gambar 8.9.

```
D:\golang\src\bab8>go run main.go
[{"nama_mahasiswa":"Aria","kursus_mahasiswa":"Funda Web"}]
```

Gambar 8.9 Output JSON

#### 8.4.2 Decode *JSON* Menjadi Suatu Object

Decode merupakan proses dimana penafsiran suatu pesan dan menterjemahkan menjadi informasi yang berarti. Dalam bahasa Golang data *JSON* bertipe data string . Dengan menggunakan fungsi “**json.Unmarshal**” , *JSON* string dapat dikonversi menjadi bentuk objek, dapat dalam bentuk variabel objek hasil struct . ,map[string]interface{}, atau slice. Berikut adalah contoh merubah suatu *JSON* menjadi suatu object slice struct, dapat dilihat pada Gambar 8.10.

```
1 package main
2
3 import (
4     "fmt"
5     "encoding/json"
6 )
7
8 func main(){
9
10     var jsonString = `[{"nama_mahasiswa" : "Aria", "kursus_mahasiswa" : "Funda Web"}]`
11     var jsonData = []byte(jsonString)
12
13     var data_mahasiswa []lepkom
14
15     var err = json.Unmarshal(jsonData, &data_mahasiswa)
16     if err != nil {
17         fmt.Println(err.Error())
18         return
19     }
20
21     fmt.Printf("%v",data_mahasiswa)
22     fmt.Printf("%s",data_mahasiswa[0].Nama)
23
24 }
25
26 type lepkom struct {
27     Nama string `json:"nama_mahasiswa"`
28     Kursus string `json:"kursus_mahasiswa"`
29 }
30
```

Gambar 8.10 Decode JSON to Object

Hasil decode nantinya akan disimpan ke variabel objek cetakan struct User. Selanjutnya siapkan data json string sederhana. Perlu dilakukan casting ke tipe



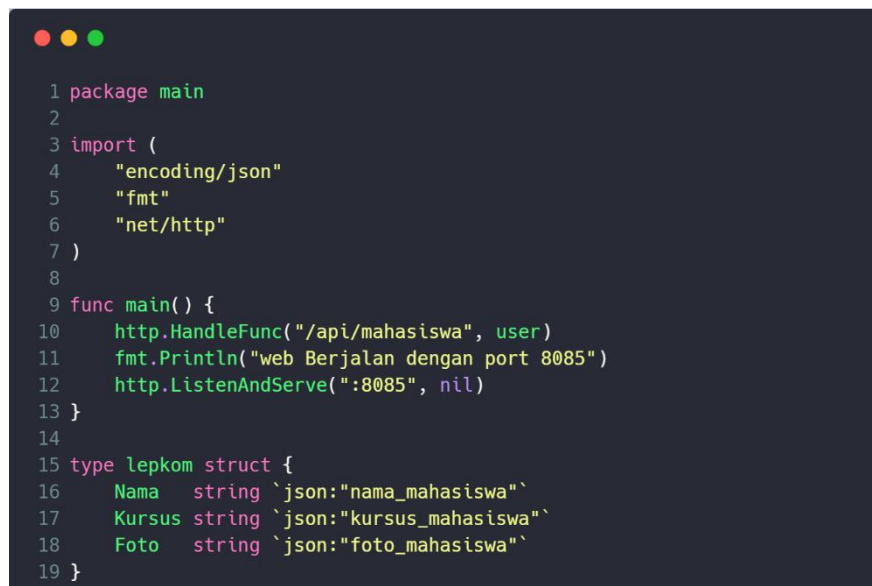
[]byte , karena fungsi `”json.Unmarshal“` hanya menerima data bertipe []byte. Dalam penggunaan fungsi `json.Unmarshal` , variabel penampung hasil decode harus di-pass dalam bentuk pointer, contohnya seperti `&data_mahasiswa`.

#### 8.4.3 Membuat API sederhana dengan format data JSON

API (Application Programming Interface) adalah sebuah teknologi untuk memfasilitasi pertukaran informasi atau data antara dua atau lebih aplikasi perangkat lunak. Sebuah API mendefinisikan bagaimana cara programmer memanfaatkan suatu fitur tertentu dari sebuah komputer . API tersedia untuk sistem windowing, sistem file, sistem basis data dan sistem jaringan.

Pada umumnya format data yang digunakan adalah XML, YAML atau JSON. Salah satu contoh API yang dimiliki oleh perusahaan startup raksasa adalah Facebook API yang dimana memiliki suatu fitur yang diberikan oleh Facebook kepada Developer aplikasi untuk mengembangkan aplikasinya yang kemudian dapat dipergunakan dalam halaman Facebook.

Berikut ini adalah contoh pembuatan API sederhana menggunakan Bahasa pemrograman golang, dapat dilihat pada Gambar 8.11.



```
1 package main
2
3 import (
4     "encoding/json"
5     "fmt"
6     "net/http"
7 )
8
9 func main() {
10     http.HandleFunc("/api/mahasiswa", user)
11     fmt.Println("web Berjalan dengan port 8085")
12     http.ListenAndServe(":8085", nil)
13 }
14
15 type lepkom struct {
16     Nama    string `json:"nama_mahasiswa"`
17     Kursus  string `json:"kursus_mahasiswa"`
18     Foto    string `json:"foto_mahasiswa"`
19 }
```

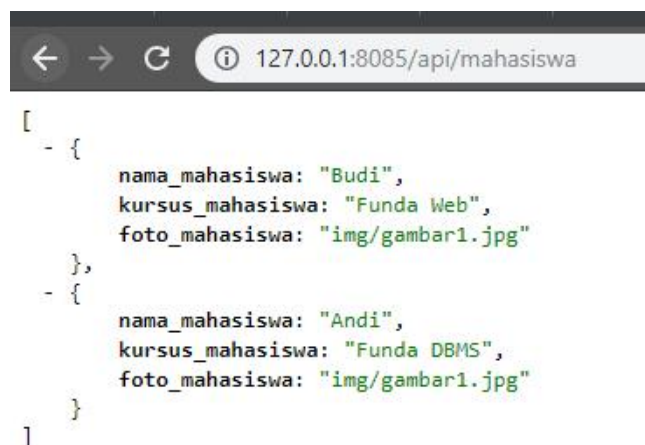
```

21 var data_mahasiswa = []lepkom{
22     {
23         Nama: "Budi",
24         Kursus: "Funda Web",
25         Foto: "img/gambar1.jpg",
26     },
27     {
28         Nama: "Andi",
29         Kursus: "Funda DBMS",
30         Foto: "img/gambar1.jpg",
31     },
32 }
33
34 func user(w http.ResponseWriter, r *http.Request) {
35     w.Header().Set("Content-type", "application/json")
36
37     if r.Method == http.MethodGet {
38
39         result, err := json.Marshal(data_mahasiswa)
40
41         if err != nil {
42             http.Error(w, err.Error(), http.StatusInternalServerError)
43             return
44         }
45
46         w.Write(result)
47         return
48     }
49 }
50

```

Gambar 8.11 API Sederhana

Pembuatannya memiliki kesamaan dengan sub bab **Encode Suatu Object Menjadi JSON**, namun kali ini tidak ditampilkan pada terminal melainkan mengeluarkan hasil json melalui URL pada web dengan cara dapat dilihat pada Gambar baris ke 10-12. Hasil yang didapatkan dapat dilihat pada Gambar 8.12.



```

[
  - {
    nama_mahasiswa: "Budi",
    kursus_mahasiswa: "Funda Web",
    foto_mahasiswa: "img/gambar1.jpg"
  },
  - {
    nama_mahasiswa: "Andi",
    kursus_mahasiswa: "Funda DBMS",
    foto_mahasiswa: "img/gambar1.jpg"
  }
]

```

Gambar 8.12 Output API

## 8.5 Web Component

*Web Component* adalah sekumpulan platform API web yang memungkinkan untuk membuat tag HTML khusus yang dapat digunakan berkali-kali, dan dienkapsulasi untuk digunakan di halaman web dan aplikasi *web*. Komponen dibuat berdasarkan standar *Komponen Web* yang didesain agar berfungsi di seluruh *web browser* modern, dan dapat digunakan dengan *library* JavaScript atau kerangka yang bekerja dengan HTML. *Web Component* didasarkan pada standar web yang ada pada saat ini. Fitur untuk mendukung *Web Component* saat ini sedang ditambahkan ke spesifikasi HTML dan DOM, membiarkan pengembang web dengan mudah memperluas HTML dengan elemen baru dengan gaya enkapsulasi dan perilaku khusus yang dimiliki.

### 8.5.2 Spesifikasi Web Component

*Komponen web didasarkan pada empat spesifikasi utama, yaitu :*

1. Custom Element

Spesifikasi *Custom Element* digunakan sebagai dasar untuk merancang dan menggunakan jenis elemen DOM baru.

2. Shadow DOM

Spesifikasi Shadow DOM mendefinisikan cara menggunakan gaya enkapsulasi dan markup dalam *Web Component*.

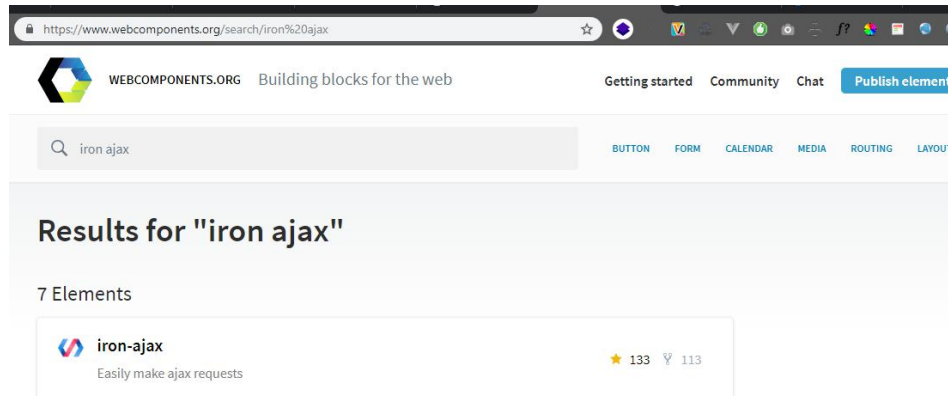
3. ES Modules

Spesifikasi ES Modules menentukan cara penggunaan kembali dokumen JS dengan berbasis standar, modular, dan kinerja dokumen js.

4. HTML Template

Spesifikasi HTML Template menentukan cara mendeklarasikan markup HTML yang tidak digunakan ketika halaman dibuka namun dapat dipakai saat Web Aplikasi berjalan dan ingin digunakan.

### 8.5.3 Penggunaan Web Component pada Polymer



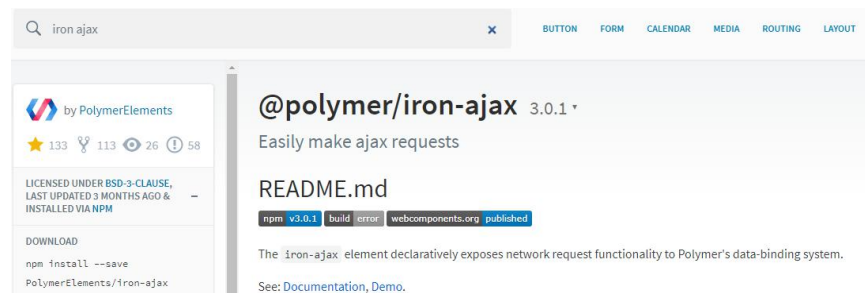
Gambar 8.13 *Web Component*

*Web Component* dapat dicari pada website [www.webcomponents.org](http://www.webcomponents.org), web tersebut menyediakan berbagai web component yang disediakan oleh pihak *official* maupun *unofficial*. Untuk melakukan instalasi web component pada website maupun aplikasi web dapat menggunakan 2 cara sesuai dengan versinya yaitu :

#### A. Menggunakan *NPM*

Sebuah tool/aplikasi kecil untuk mengatur package/aplikasi JavaScript yang menggunakan Node.js, dibawah ini merupakan contoh instalasi web component iron-ajax menggunakan NPM

1. Ketahui komponen apa yang ingin digunakan pada website web component dan cari komponen dengan mengetikan nama komponen pada kolom pencarian.



Gambar 8.14 *Web Component iron-ajax*

2. Disebelah kiri terdapat tulisan download, copy tulisan tersebut pada terminal/ command prompt yang digunakan pada komputer anda, pastikan komputer sudah terinstall Node.js dan pastikan direktori pada terminal sudah berada di direktori project polymer yang dibuat.

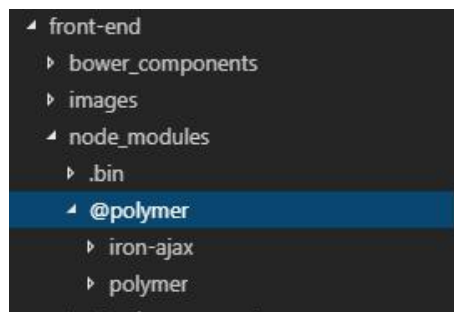
```
C:\golang\src\pertemuan8\front-end>npm install --save @polymer/iron-ajax
npm WARN polymer-starter-kit@ No repository field.

+ @polymer/iron-ajax@3.0.1
added 3 packages from 3 contributors and audited 414 packages in 6.279s
found 0 vulnerabilities

C:\golang\src\pertemuan8\front-end>
```

Gambar 8.15 Instalasi dengan NPM

Maka pada project polymer didalam folder node\_modules/@polymer sudah terdapat iron ajax



Gambar 8.16 Struktur Folder

## B. Menggunakan *Bower*

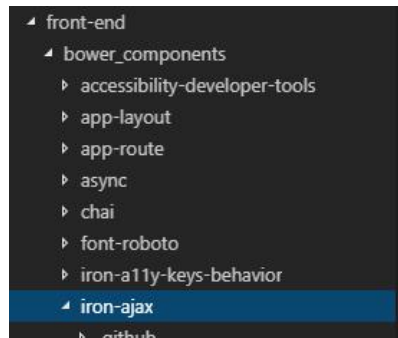
Bower adalah salah satu package manager untuk pengembangan front-end web. Bower berfungsi untuk memudahkan kita mengatur paket atau library apa saja yang kita butuhkan saat mendvelop suatu aplikasi, dalam hal ini front-end. Untuk instalasi menggunakan bower sama seperti menggunakan NPM, namun hanya saja berbeda sintaks pada terminal, seperti dibawah ini.

```
C:\golang\src\pertemuan8\front-end>bower install --save PolymerElements/iron-ajax
bower iron-ajax#* cached https://github.com/PolymerElements/iron-ajax.git#3.0.1
bower iron-ajax#* validate 3.0.1 against https://github.com/PolymerElements/iron-ajax.git#*
bower iron-ajax#*3.0.1 install iron-ajax#3.0.1

iron-ajax#3.0.1 bower_components\iron-ajax
C:\golang\src\pertemuan8\front-end>
```

Gambar 8.17 Instalasi dengan Bower

Maka pada project polymer didalam folder bower\_components/ sudah terdapat iron ajax



Gambar 8.18 Struktur folder bower

Setelah melakukan instalasi, web component dapat langsung digunakan namun pertama-tama harus terlebih dahulu didaftarkan pada halaman HTML yang ingin menggunakan komponen tersebut, seperti gambar dibawah ini.

```
1 <link rel="import" href="../../bower_components/polymer/polymer-element.html">
2 <link rel="import" href="../../bower_components/iron-ajax/iron-ajax.html">
3 <link rel="import" href="shared-styles.html">
4
```

Gambar 8.19 Mendaftarkan *Web Component*

Setelah komponen didaftarkan, komponen tersebut dapat langsung digunakan dengan menggunakan tag khusus sesuai dengan komponen yang digunakan, dalam contoh kali ini komponen yang digunakan adalah iron-ajax maka tag khususnya adalah `<iron-ajax></iron-ajax>`, dapat dilihat pada Gambar

```
1 <iron-ajax
2   auto
3   url="/api/mahasiswa"
4   handle-as = "json"
5   mehotd = "GET"
6   on-response="_handleResponse"
7   debounce-duration="300">
8 </iron-ajax>
```

Gambar 8.20 Menggunakan *Web Component*

Untuk komponen iron-ajax perlu beberapa tambahan kode pada script untuk menampung data (variable data) dan mengolah data (fungsi `_handleresponse`) tersebut, dapat dilihat pada Gambar

```

1  <script>
2      var data;
3      class MyView1 extends Polymer.Element {
4          static get is() { return 'my-view1'; }
5
6          // Variable untuk menampung Data
7          static get properties(){
8              return {
9                  data : {
10                      type : Object,
11                      notify : true,
12                  },
13              }
14          }
15          // Fungsi untuk megolah data
16          _handleResponse(e){ this.data = e.detail.response;
17      }
18
19      window.customElements.define(MyView1.is, MyView1);
20  </script>

```

Gambar 8.21 Variabel dan Fungsi

Setelah data didapatkan dari iron-ajax, maka data tersebut dapat diolah sesuai keinginan developer, sebagai contoh dibawah ini data dari url /api/mahasiswa yang diambil menggunakan iron-ajax diolah dan ditampilkan pada web browser dengan menggunakan fungsi bawaan dari polymer yaitu dom-repeat untuk mengolah data dan ditampilkan dalam img serta paragraph, seperti Gambar 8.22.

```
<center><h1>Daftar Nama Mahasiswa Kursus Lepkom </h1></center>  
<template is="dom-repeat" items="[[data]]">  
    <div class="card">  
          
        <p>Nama   &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;& [[item.nama_mahasiswa]]</p>  
        <p>Kursus :&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;& [[item.kursus_mahasiswa]]</p>  
    </div>  
</template>
```

Gambar 8.22 Fungsi dom-repeat

Maka kode lengkap dari program sederhana menggunakan iron-ajax, dapat dilihat pada Gambar 8.23. **\*Project polymer yang digunakan adalah project polymer 2 starterkit, untuk modifikasi dilakukan pada myview1.html pada folder src. \***

```

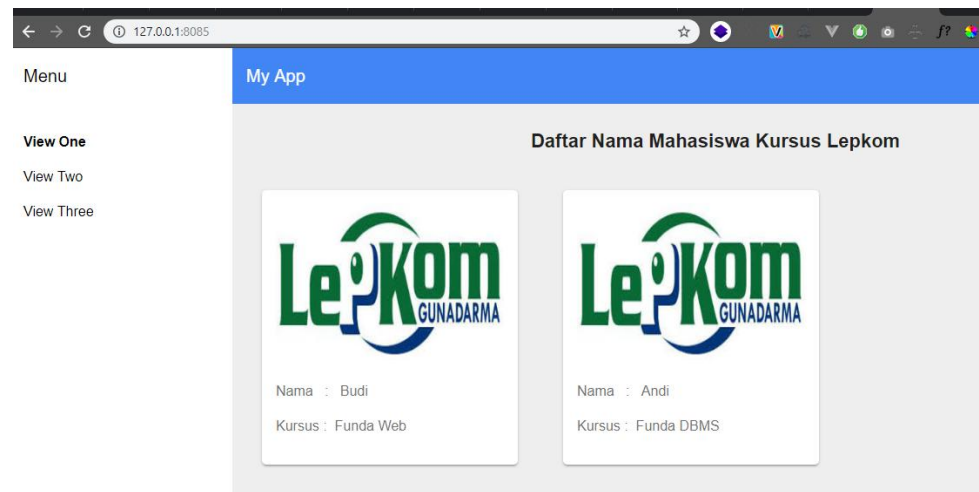
1 <link rel="import" href="../../../bower_components/polymer/polymer-element.html">
2 <link rel="import" href="../../../bower_components/iron-ajax/iron-ajax.html">
3 <link rel="import" href="shared-styles.html">
4
5 <dom-module id="my-view1">
6   <template>
7     <style include="shared-styles">
8       :host {
9         display: block;
10
11         padding: 10px;
12       }
13     </style>
14     <custom-style>
15       <style is="custom-style">
16         .card {
17           width : 24%;
18           display: inline-block;
19         }
20         .gambar {
21           width: 100%;
22           height: 180px;
23         }
24       </style>
25     </custom-style>
26     <center> <h1> Daftar Nama Mahasiswa Kursus Lepkom </h1></center>
27     <template is="dom-repeat" items="{{[data]}}">
28       <div class="card">
29         
30         <p>Nama      &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&{{item.nama_mahasiswa}}</p>
31         <p>Kursus  :&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&{{item.kursus_mahasiswa}}</p>
32       </div>
33     </template>
34
35     <iron-ajax
36       auto
37       url="/api/mahasiswa"
38       handle-as = "json"
39       method = "GET"
40       on-response="*_handleResponse"
41       debounce-duration="300">
42     </iron-ajax>
43   </template>
44   <script>
45     var data;
46     class MyView1 extends Polymer.Element {
47       static get is() { return 'my-view1'; }
48
49       static get properties(){
50         return {
51           data : {
52             type : Object,
53             notify : true,
54           },
55         }
56       }
57       _handleResponse(e){ this.data = e.detail.response}
58     }
59
60     window.customElements.define(MyView1.is, MyView1);
61   </script>
62 </dom-module>
63

```

Gambar 8.23 Kode Lengkap Project



Output pada program diatas dapat dilihat pada Gambar 8.24.



Gambar 8.24 Output Program