Soal terkait

1. **SISTER.Js** (6 poin + 9 poin bonus)

*Source gambar:* [*https://plainenglish.io/blog/top-10-javascript-frameworks-for-server-side-development-in-2020-6d265016c02*](https://plainenglish.io/blog/top-10-javascript-frameworks-for-server-side-development-in-2020-6d265016c02)

A collage of a logo

Description automatically generated with medium confidence

Let’s add more !!!

Ketentuan

Buatlah *framework backend* dengan memanfaatkan **socket** dengan menggunakan  **NodeJs,** **Python** , **Java** , **Golang , C , C++**, atau **Rust**. Spesifikasi wajib *framework* adalah sebagai berikut:

* Melakukan *routing* *request* berdasarkan url dan *request method*. Contoh:

|  |
| --- |
| app.get(‘/nilai-akhir’, NilaiController.getNilaiAkhir) |

Kode ini setiap kali ada request masuk dengan url ‘[http://[IP]:[PORT]/nilai-akhir](about:blank)’ dan request method **GET**, maka akan mengarahkan request ke method  **NilaiController.getNilaiAkhir** untuk kemudian diproses oleh method tersebut.

* Menerima request **GET, POST, PUT,** dan **DELETE** dengan spesifikasi minimal setiap *request method* sebagai berikut:

1. **GET**:
   1. Bisa melakukan routing url biasa(ex: [www.google.com](http://www.google.com) )
   2. Bisa melakukan routing url yang  mengandung query param beserta menyediakan method untuk meng-ekstrak query paramnya (ex: [www.google.com?q=123](http://www.google.com?q=123) )
2. **POST**:
   1. Bisa melakukan routing url biasa (ex: <http://recruit.sister20.tech/submit/a>)
   2. Bisa memproses request body. Jenis content-type minimal yang bisa diproses antara lain:
      1. text/plain
      2. application/json
      3. application/x-www-form-urlencoded
3. **PUT**:
   1. Bisa memproses url biasa(ex: [www.google.com](http://www.google.com) )
   2. Bisa memproses request body. Jenis content-type minimal yang bisa diproses antara lain:
      1. text/plain
      2. application/json
      3. application/x-www-form-urlencoded
4. **DELETE**:
   1. Bisa memproses url biasa(ex: [www.google.com](http://www.google.com) )

* Dapat memproses HTTP Header. Header minimal yang bisa diproses antara lain:
  + Content-Type
  + Accept
* Mengirimkan response kepada *client* beserta response code yang sesuai dalam bentuk **plaintext** atau **JSON** yang di-*stringify* (Untuk parsing/stringify JSON boleh menggunakan *library* atau membuat sendiri jika ingin mendapatkan nilai bonus)

Bonus

Terdapat beberapa fitur tambahan yang bisa kamu selesaikan jika ingin nilai bonus (+ lagi gabut) sebagai berikut:

1. (Poin 1) Handling *middleware*.
2. (Poin 1) Parsing Content-Type: multipart/form-data
3. (Poin 1) Membuat *parser* JSON tanpa bantuan library (Silahkan implementasikan ilmu TBFO dan Stima)
4. (Poin 2) Handle *file upload* serta dapat mengirimkan *response* berupa *file* (Hint: Implementasikan parser **multipart/form-data**)
5. (Poin 4) Paralelisasi penerimaan dan pemrosesan *request* yang masuk

Tujuan

* Mempelajari bagaimana *framework* yang biasa dipakai di *backend* bekerja
* Menghubungkan materi *low level* yang dipelajari di Jarkom dengan high level di *webdev*
* Kalau ditanya dosen , bisa pamer kalau pernah bikin *framework* BE *from scratch* ( fun fact di kelas WBD tahun lalu, Pak Yudis nanya di kelas apakah ada yang pernah pakai node.js buat backend **tanpa** menggunakan Express.js atau framework apapun)

Berkas

* Source code program
* Dokumen (Markdown/PDF) berisi fitur-fitur yang diimplementasikan,  cara kerjanya secara singkat, serta petunjuk cara menggunakan fitur tersebut. Untuk tiap fitur, minimal terdapat 1 gambar, baik screenshot kode atau screenshot saat sedang menjalankan fitur tersebut.

Referensi

* Referensi library socket untuk tiap bahasa:
  + <https://nodejs.org/api/net.html#net>
  + <https://docs.python.org/3/library/socket.html>
  + <https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html>
  + <https://doc.rust-lang.org/std/net/index.html>
  + <https://pkg.go.dev/net>
  + <https://www.gnu.org/software/libc/manual/html_node/Sockets.html>
  + <https://man7.org/linux/man-pages/man2/socket.2.html>
* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>
* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>
* <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST>

## Latar Belakang

Oke jadi buat masalah ini penulis bikin framework make bahasa piton (karena gampang, duh). Untuk desainnya penulis niru banyak banget dari flask. Sebelumnya penulis belom pernah bikin web (ril gw baru belajar bikin web langsung bikin framework), jadi struktur data dan penggunaan dari framework yang penulis buat mungkin cukup aneh, mudah – mudahan karena niru flask jadinya ga begitu aneh yak.

Sekilas dulu ini struktur file dari sisterjs

A screenshot of a computer

Description automatically generated

Buat asset, script, data, sama file - file html itu sebenernya dummy semua kwkwkw. Yang penting semuanya di lib (kecuali client\_dummy sama database itu buat testing juga).

Untuk demonya bisa coba jalanin main.py. Di main.py juga ada comment tentang fitur – fitur yang ada di sini (siapa tau males baca). terus client dummy untuk testing kiriman request HTTP atau buka localhost:10000/ di browser, ntar ada web dummy yang isinya crud database.

Untuk implementasinya penulis juga buat dengan object oriented, dengan kelas Server sebagai kelas utamanya dan Server\_Handler itu worker thread buat servernya karena bikin multithreading, kelas Request untuk representasi request dari user dan kelas Response untuk membuat response HTTP dari bentuk yang diinput. Util isinya fungsi statik tambahan atau konstanta.

## Fitur yang diimplementasikan

* + 1. Routing request

Untuk pertama yang jelas bikin routing, buat routingnya pertama buka socket dulu di port yang diinginkan, ini ada di kelas server. Port sama address bisa dipilih tapi defaultnya kek di bawah

A screen shot of a computer program

Description automatically generated

Terus untuk bikin rutenya, penulis niru flask dengan bikin decorated function, penggunaannya kek gini

A black screen with green and yellow text

Description automatically generated

Kalo diliat fungsi route di sana banyak yang bakal ngebingungin soalnya penulis rada ngide buat bisa ngeroute dengan argument di urlnya, bukan kaya query, tapi kaya gini

A screen shot of a computer program

Description automatically generated

Untuk implementasinya sendiri itu berupa dictionary of string to function, di Server ada di self.routes. Kalo buat yang argument dari url, itu ada routes\_vars, isinya nyimpen variabel ada di input ke berapa (ngitung ‘/’) terus tipenya apa. Buat yang disimpen di self.routes untuk yang variabel dia tetep Namanya literal ‘/api/dummydata/<int>/uhh/<str>’. Nanti waktu Nerima respons aja diparse buat integer sama stringnya diterima sebagai argument terus diganti jadi <int> sama <str>

Buat nyari responsenya itu ada di fungsi server.response(request\_data: str). Request\_data diproses dulu jadi kelas Request yang lebih rapih dan ngambil ngambilin info kek tipe request, address, extract querynya, sama extract content yang dikirim. Request juga udah ngambil header Accept, Content-Type, sama Content-Length, kek gini doang kelasnya:

A screen shot of a computer program

Description automatically generated

Query bakal dipisahin dari address route, spasi dipake buat parsing sama, tanda tanya harus ada di terakhir. Karena itu gaboleh ada rute yang make tanda tanya sama gaboleh ada rute yang make spasi. Parsing query make ini:

A computer screen shot of a code

Description automatically generated

Fungsi ini juga dipake buat proses data yang bentuknya Application/x-www-form-urlencoded

Habis itu dicek ada routenya atau engga, awalnya cek polosan, kalo gaada bakal dicek bisa dari route yang punya variabel atau engga. Beres itu dicek lagi di rute itu ada metode yang dibolehin atau engga. Kalo ada baru diproses responsenya sesuai fungsi dekorator yang ditambahin.

Buat return fungsi dekoratornya harus berupa Response, kecuali string, return berupa string bisa otomatis diubah jadi Response. Untuk html sama json ada fungsi wrappernya buat response yaitu json\_response(data: dict) sama html\_response(path: str). Di kelas response, dari input yang dibuat bisa digenerate data yang perlu disend, kelasnya kek gini:

A screen shot of a computer screen

Description automatically generated

Generate itu buat bikin data yang perlu disend, override itu buat ngeganti beberapa data sebelum degenerate, awal kepikiran use casenya kalo error biar ga bikin kelas baru tapi gatau deng guna atau engga

Habis responsenya di generate, bakal disend sama Server\_Handler ke klien. Kurang lebih intinya gitu sih

* + 1. GET request

Basic aja, GET bakal routing sama ngirim data balik di Response kaya yang udah dijelasin sebelumnya. Paling tambahan, kalo ga dispesifikasi di dekoratornya method apa, dia bakal default ke ‘GET’ doang. Maksudnya kek gini:

A screen shot of a computer

Description automatically generated

* + 1. POST request

Buat POST, routing sama seperti yang udah dijelasin sebelumnya juga. Buat proses body itu ada dari ctor-nya request. Karena penulis ga bikin yang kirim kiriman data, yang bisa diterima cuman Application/json, Application/x-www-form-urlencoded, sama text/plain. Application/json sama Application/x-www-form-urlencoded selalu diubah ke bentuk dictionary. text/plain sebagai string. Ini buat fungsi wrappernya:

A screen shot of a computer code

Description automatically generated

extract\_plaintext ada biar keren aja

* + 1. PUT request

Buat PUT, routing sama seperti yang udah dijelasin sebelumnya juga. Buat ekstraksi content body juga sama persis kayak di POST request.

* + 1. DELETE request

Buat DELETE, routing sama seperti yang udah dijelasin sebelumnya juga. Gaada khusus ngapa -ngapain, bikin fungsi decoratornya aja sendiri.

* + 1. HTTP Header

Extract HTTP Header diproses sama kelas request dengan cara yang jelek banget sebenernya, cuman jalan jadi yaudah dah males mikir dan skil isu soalnya penulis

A screen shot of a computer program

Description automatically generated

Yah gitu caranya, linear terus pake str.find dari piton

* + 1. Response

Untuk response sebenernya kurang lebih cuman bisa text sama json (ada webp juga sih yang ngirim byte). Penulis ga bikin parser sendiri dan make library json. Kaya udah dijelasin pas awal, ngirimnya make kelas Response buat ngeencode jadi paket HTTP yang sesuai. Cuman buat html sama json wrapper di response

A screen shot of a computer program

Description automatically generated

Ini create\_json di util

A black screen with white text

Description automatically generated

Gitu doang, gak begitu kompleks soalnya penulis ga bikin yang kirim kiriman file

* + 1. Middleware

Penulis bikin support middleware, lagi lagi kaya flask, make decorator dan ada dua versi middleware, yaitu before\_request sama after\_request

A screen shot of a computer

Description automatically generated

Buat cara kerjanya sederhana banget, setiap fungsi yang ditambah ke decorator bakal diappend ke array of functions di server

A black background with white text

Description automatically generated

Terus semua fungsi yang ada di dalem array itu selalu dijalanin setiap sebelum atau sesudah request sesuai dekorator yang dipake

A screen shot of a computer program

Description automatically generated

Ntar yang ngejalanin bukan server tapi worker thread, ada di poin fitur selanjutnya

* + 1. Multithreading

Kaya udah penulis singgung di awal, penulis make multithreading buat handle client make library Threading dari python. Strukturnya berupa worker thread gitu di kelas Server\_Handler. Ini dia make dependency injection buat ngejalanin fungsi yang ada sama akses data yang perlu dari servernya

A computer screen shot of a program

Description automatically generated

Jadi kalo ada koneksi, server cuman perlu bikin worker thread ini terus jalanin

A screen shot of a computer code

Description automatically generated

* + 1. Static Loading

Isi folder bisa diload secara custom buat masukin folder apa aja yang bakal bisa di fetch pake method GET, jadi ga cuman di folder static.

A black background with white text

Description automatically generated

Jadi kalo mau masukin script .js atau asset foto foto bisa langsung terus bisa distrukturin sendiri karena udah rekursif ke folder dalem foldernya juga.

Cuman kebetulan karena penulis skill isu, jadi file yang bisa ada di folder – folder itu cuman .html, .css, .js, .webp, .json, .txt (Penulis ga implementasiin yang kirim – kiriman file hehe)

* + 1. Unconventional Requests

Ya karena ga di spesifikasiin methodnya apa aja ya di Server sama di Responses, sebenernya secara teori kalo mau bikin method request yang ngarang gitu misal HTTP requestnya mau handle make request “WOKWOKWOKWOK”, itu juga bisa. Asal dari frontend atau dari yang ngirim requestnya mau ngirim request itu. Kebetulan penulis belom nyoba yang ini sih (penulis males), cuman secara teori harusnya bisa.

## Penutupan

Yah jadi ini soal yang lumayan menarik penulis ngerjainnya. Kapan lagi belajar web dengan bikin framework ya ga ya ga. Tengkyu udah baca pseudo-laporan ini.