JINTEKS (Jurnal Informatika Teknologi dan Sains)
Vol. 4 No. 1, Februari 2022, hlm. 8 – 14

ISSN 2686-3359 (Online)
DOI Prefix 10.51401

RANCANG BANGUN *REST API* APLIKASI *WESHARE* SEBAGAI UPAYA MEMPERMUDAH PELAYANAN DONASI KEMANUSIAAN

Hasanuddin[1*], Hari Asgar[2], Budi Hartono[3]
[1), 2), 3)] Teknik Informatika, Fakultas Teknik, Universitas Cordova
email: hasanuddin@undova.ac.id[*]

Abstrak: Kegiatan donasi secara umum dilakukan melalui media *offline* atau langsung yang tentunya mempersulit masyarakat untuk melakukan transaksi donasi, karena harus datanh langsung menuju lokasi pembukaan layanan donasi, selain itu juga jangkauan pemberi donasi yang terbatas jika dilakukan secara offline. Dari permasalahan tersebut, dikembangkan sistem transaksi donasi berbasis *Application Programming Interface (API)* sebagai *Backend Development* dan diimplementasikan untuk *Mobile* dan *Website* berbasis Android. Penelitian ini menghasilkan sistem berbasis *API* dengan arsitektur *REST* dalam hal *Backend Development* untuk memudahkan masyarakat dalam memberikan transaksi donasi dan diterapkan pada aplikasi Android dan *website* sebagai *user interface*.

Kata Kunci : *API, Backend Development, Donation Transactions, REST API*

Abstract: *Donation activities are generally carried out through offline or direct media which of course makes it difficult for people to make donation transactions, because they have to come directly to the location of the donation service opening, besides that the reach of donors is limited if done offline. From these problems, an Application Programming Interface (API)-based donation transaction system was developed as Backend Development and implemented for Android-based Mobile and Websites. This research produces an API-based system with a REST architecture in terms of Backend Development to make it easier for the public to provide donation transactions and is applied to Android applications and websites as a user interface.*

Keywords **:** *API, Backend Development, Donation Transactions, REST API*

PENDAHULUAN

Pada umumnya pelayanan donasi hanya dilakukan secara langsung atau melalui perantara yang tentunya akan mempersulit pelaksanaan kegiatan donasi, karena harus mengeluarkan tenaga untuk ke lokasi pelayanan donasi, juga dilihat dari efisiensi waktu dan biaya yang dikeluarkan oleh donatur. Dengan berkembangnya teknologi informasi saat ini, banyak ide atau gagasan baru terkait layanan donasi yang bisa dilakukan secara *online*. Dimana donatur dapat langsung melihat kategori dan formulir donasi yang ditawarkan. Jadi donatur bisa memproses donasi dari mana saja. Untuk mengatasi masalah tersebut maka perlu dirancang sistem layanan donasi berbasis *Application Programming Interface (API)* sebagai *Backend Development* dan diimplementasikan pada *website* dan *mobile* berbasis Android. *Backend Development* itu sendiri bertanggung jawab untuk sisi *server* dan *database*. Dengan demikian peneliti merancang *Representatif State Transition Application Programming Interface (REST API)*. *REST* sendiri merupakan standar arsitektur komunikasi yang biasa diterapkan dalam pengembangan situs *website* dan layanan berbasis aplikasi, sedangkan *API* adalah tautan yang memungkinkan aplikasi untuk berinteraksi dan berbagi data.

Application Programming Interface (API) adalah antarmuka yang dibangun oleh pengembang sistem sehingga beberapa atau semua fungsi sistem dapat diakses secara terprogram. Peneliti menunjukkan bahwa pengembangan *API* telah berhasil, dan implementasi *REST* telah mempermudah pengembangan struktur *API*. Dengan bantuan *API*, *user* dapat *register* atau *login* ke aplikasi hanya dengan mengirimkan *email* ke aplikasi dan memudahkan proses transaksi donasi. *API* juga dapat berperan dalam memudahkan *user* melakukan berbagai metode pembayaran untuk membuat proses *checkout* menjadi lebih cepat dan mudah.

METODE

Metode yang digunakan dalam perancangan *REST API* yaitu dengan menggunakan metode SCRUM yaitu salah satu jenis metode perancangan *System Development Life Cycle (SDLC)*, yang mempunyai beberapa tahapan diantaranya :

1. Identifikasi

Membutuhkan waktu yang lebih dikarenakan peneliti mencari segala jenis *website* untuk mendapatkan informasi terkait data pelayanan donasi untuk dijadikan sebagai referensi.

2. Analisis

Pada tahapan ini dilakukan pengumpulan data dan informasi yang berhubungan dengan data apa saja yang harus ditampilkan oleh *Backend* melalui analisis Figma *design* yang disediakan oleh : https://www.figma.com/file/JkmuHtkou4arhgNWY7fBgU/WeShare?node-id=15%3A1582 sebagai acuan utama dalam analisis data.

ISSN 2686-3359 (Online)
DOI Prefix 10.51401

3. Perancangan

Peneliti membuat arsitektur sistem, untuk tahap perancangan data yang digunakan adalah data *dummy* untuk melakukan testing sending data API berbentuk JSON terhadap pihak *Frontend* dengan menggunakan bahasa pemrograman JavaScript, NodeJS dan ExpressJS sebagai *framework* dalam perancangan *REST API*.

4. Implementasi

Spesifikasi perangkat keras dan perangkat lunak yang digunakan peneliti dalam pembuatan aplikasi sebagai berikut : Satu unit laptop dengan *processor AMD Athlon Silver*. *RAM* dengan tipe DDR4 berkapasitas 8GB. WD SSD berkapasitas 512GB. *Graphics Card AMD Radeon*. Sedangkan dalam mendesain skema *Database* menggunakan Draw.io, *Visual Studio Code* dengan ekstensi *Live Server*, GIT sebagai *Version Control System*, *Web browser* yang bernama Chrome, Postman untuk mempermudah *testing REST API* dan tipe *database* PostgreSQL dengan bantuan Sequelize untuk mempermudah lalu lintas data ke dalam *database*.

5. Uji Coba/*Testing*

Melakukan uji coba terhadap masing-masing REST API yang telah dibuat pada situs *URL* (*Uniform Resource Locator*) website menggunakan bantuan Postman. Dilakukannya uji coba ini supaya *API* dapat berjalan menghantarkan data secara optimal sebagaimana mestinya.

HASIL DAN PEMBAHASAN

Pembuatan aplikasi dirancang sedemikian rupa, hal ini berguna supaya pada saat perancangan *REST API* tersebut tidak ada tahap yang terlewati sehingga membuat minimnya terjadi kesalahan,dan hasil *output* nantinya sesuai dengan pemecahan masalah yang ada. Peneliti membagi proses perancangan *REST API* yang terimplementasi dalam aplikasi WeShare ini menjadi 3 (tiga) bagian, yaitu perancangan *Backend Development*, perancangan arsitektur *system* dan perancangan *API*.

Dalam perancangan *REST API* yang digunakan dalam aplikasi *WeShare* ini tentu terdapat beberapa proses perancangan sehingga menghasilkan *REST API* yang terimplementasi dalam aplikasi *WeShare*. Adapun proses perancangan yang ditempuh *Backend Development* dalam tim WeShare ini adalah sebagai berikut :

1. Perancangan *Backend Development*

Backend Development adalah proses perancangan *system* yang dilakukan di belakang layar dari sebuah *website* ataupun aplikasi. Bahasa pemograman untuk *Backend Development* diantaranya adalah Javascript, PHP, Ruby, Python, dan banyak lainnya. Itu semua adalah mesin yang bekerja dibalik layer dan tentu tidak terlihat secara langsung oleh pengguna, akan tetapi dalam sebuah perancangan sebuah website ataupun aplikasi posisi *Backend Development* memberikan kekuatan pada apa yang terjadi terutama dalam pengolahan data yang akan ditampilkan. *Backend Development* fokus pada *database*, *scripting*, dan arsitektur dari sebuah *website* atau aplikasi. Kode yang ditulis oleh *Backend* akan membantu mengomunikasi informasi data yang terdapat pada *database* kepada user melalui perantara tampilan *User Interface* (*UI*) yang disediakan oleh *Frontend*. Berikut beberapa alat Bantu Pemrograman Backend Development yang digunakan dalam perancangan *REST API* :

a. Application Programming Interface (API)

API adalah antarmuka yang digunakan untuk mengakses aplikasi atau layanan dari sebuah program. *API* memungkinkan pengembang untuk memakai fungsi yang sudah ada dari aplikasi lain sehingga tidak perlu membuat ulang dari awal. Pada konteks *website*, API merupakan pemanggilan fungsi melalui *Hyper Text Transfer Protocol (HTTP)* dan mendapatkan respon berupa *Extensible Markup Language (XML)* atau *JavaScript Object Notation (JSON)*.

Tujuan penggunaan dari *API* adalah untuk saling berbagi data antar aplikasi yang berbeda, Tujuan penggunaan *API* lainnya yaitu untuk mempercepat proses pengembangan aplikasi dengan cara menyediakan sebuah *function* yang terpisah sehingga *developer* tidak perlu lagi merancang fitur yang serupa. *API* yang bekerja pada tingkat sistem operasi membantu aplikasi berkomunikasi dengan *layer* dasar dan satu sama lain mengikuti serangkaian protokol dan spesifikasi yang telah disesuaikan.

b. Representional State Transfer (REST)

REST merupakan seperangkat prinsip arsitektur yang melakukan transmisi data melalui antarmuka yang terstandarisasi seperti *HTTP*. *REST* bekerja layaknya seperti aplikasi *website* biasa. *Client* dapat mengirimkan permintaan kepada *server* melalui protokol *HTTP* dan kemudian *server* memberikan *respons* balik kepada *client*. *REST* dikembangkan oleh Roy Fielding yang merupakan *co-founder* dari *Apache HTTP Server Project.*

Pada arsitektur *REST* itu sendiri, *REST server* menyediakan *resources* (sumber daya/data) dan *REST client* mengakses dan menampilkan *resource* tersebut untuk penggunaan selanjutnya. Setiap *resource* diidentifikasi oleh *URIs (Universal Resource Identifiers)* atau *global ID*. *Resource* tersebut direpresentasikan dalam bentuk format teks, *JSON* atau *XML*.

c. JavaScript Object Notation (JSON)

JSON adalah sebuah format untuk berbagi data. Sesuai dengan namanya, *JSON* diturunkan dari bahasa pemrograman Javascript, akan tetapi format

ISSN 2686-3359 (Online)
DOI Prefix 10.51401

ini tersedia bagi banyak bahasa lain termasuk Python, Ruby, PHP, dan Java. *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 *client* atau browser.

d. NodeJS

NodeJS adalah perangkat lunak yang didesain untuk mengembangkan aplikasi berbasis *web* dan ditulis dalam sintaks bahasa pemrograman Javascript. Bila selama ini kita mengenal Javascript sebagai bahasa pemrograman yang berjalan disisi *client/ browser* saja, maka *NodeJS* ada untuk melengkapi peran JavaScript sehingga bisa juga berlaku sebagai bahasa pemrograman yang berjalan disisi *server*, seperti halnya PHP, Ruby, Perl, dan sebagainya. *NodeJS* dapat berjalan disistem operasi Windows, Mac OS dan Linux tanpa perlu ada perubahan kode program. *NodeJS* memiliki pustaka *server HTTP* sendiri sehingga memungkinkan untuk menjalankan *server web* tanpa menggunakan program *server web* seperti Apache atau Nginx.

e. Sequelize

Sequelize adalah *Object Relational Mapping (ORM)* NodeJS yang berbasis *promise*. Ini bisa digunakan dengan PostgreSQL, MySQL, MariaDB, SQLite, dan MSSQL. Fitur ini mendukung transaksi yang solid, hubungan, *lazy loading* dan *eager*, membaca replikasi dan banyak lagi.

f. JSON Web Token (JWT)

JWT ini adalah sebuah *token* berbentuk *string* panjang yang sangat *random* yang gunanya sendiri untuk melakukan sistem Autentikasi dan Pertukaran Informasi. Umumnya untuk melakukan *login* tidak seperti pada aplikasi *website* biasa dimana kita menggunakan *session* untuk mengingat siapa yang sedang *login*. Tapi didalam API sendiri kita menggunakan konsep *JWT* . *Website* resminya dapat diakses di https://jwt.io/

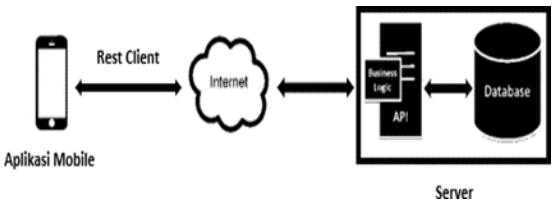
JWT terdiri dari tiga struktur yang dipisahkan oleh tanda titik (.), yaitu:

1. *Header* untuk memuat jenis *encoding* yang digunakan.
2. *Payload* untuk memuat nilai-nilai informasi yang ditransaksikan.
3. *Signature* untuk memuat nilai *hash* untuk memverifikasi *payload*

g. Dotenv

File.env (dotenv) berfungsi untuk menyimpan variabel env, file ini berisi deklarasi atau pembuatan variabel env yang nantinya dapat kita load dari *NodeJS*. Agar dapat menggunakan file .env, kita membutuhkan modul dotenv. Modul ini nanti akan membantu kita untuk me-*load* semua variabel yang ada di file .env.

2. Perancangan Arsitektur Sistem



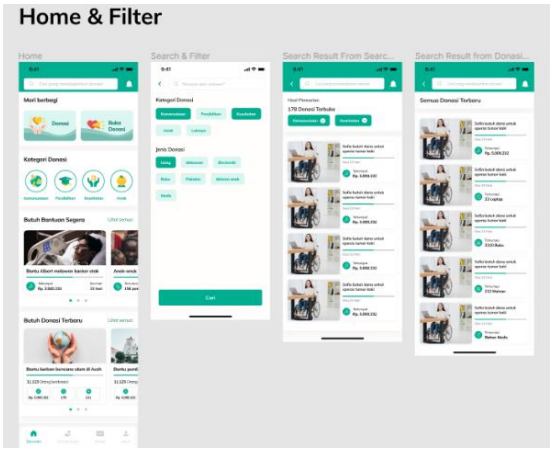
Gambar 1. Arsitektur Sistem

Aplikasi *client* (*android* dan *website*) yang di bangun pada penelitian ini terintegrasi oleh *Application Programming Interface* yang menggunakan metode *REST*. Dapat dilihat pada gambar 1 merupakan skema pengambilan data menggunakan *REST API*. Data Donasi pengguna nantinya akan digabungkan dan dikirim dalam bentuk *JSON*. Kemudian data tersebut akan dikirim ke *server API* selanjutnya *Server API* akan melakukan proses permintaan ke *database*.

Berikut tahapan-tahapan perancangan *REST API* dalam aplikasi *WeShare* :

a. Analysis API

Dalam perancangan *REST API* yang terimplementasi dalam aplikasi WeShare ini tahap pertama yang dilakukan tim *Backend* adalah menganalisis *API* apa saja yang dibutuhkan melalui *Design* gambar *MockUp UI* yang telah disiapkan. Dengan adanya sampel design *MockUp UI* yang telah disediakan maka mempermudah tim *Backend* untuk menganalisis *API* dan data apa saja yang perlu disediakan dalam masing-masing *API*.



Gambar 2. *MockUp UI Home and Filter WeShare*

Untuk *Design MockUp UI* aplikasi *WeShare* lebih lengkapnya bisa diakses melalui link berikut :

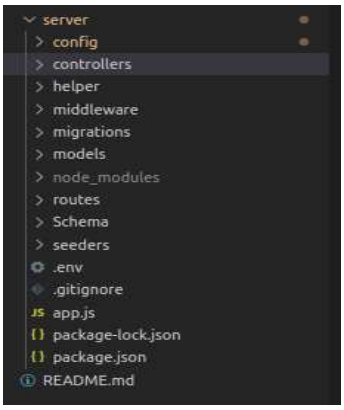
ISSN 2686-3359 (Online)
DOI Prefix 10.51401

https://www.figma.com/file/JkmuHtkou4arhgNWY7fBgU/WeShare?node-id=15%3A1582

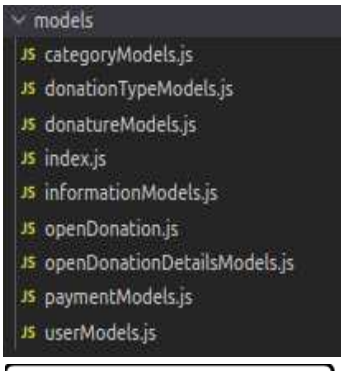
b. Design Coding Foundation

Sebuah pondasi *coding* sangat penting untuk diperhatikan, dalam perancangan *REST API* karena sangat berpengaruh untuk kelancaran *development* dalam tim termasuk untuk menghindari fatalnya *merge* pada saat *push REST API* ke *cloud server* karena terdapat perbedaan susunan folder didalam *server* yang dirancang oleh masing-masing anggota *Backend* sehingga diharuskan setiap anggota menggunakan satu pondasi *coding*.

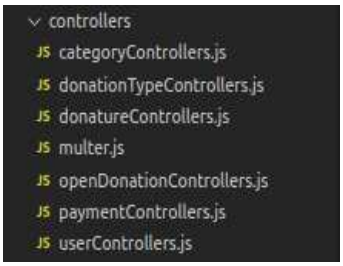
Coding Foundation yang digunakan dalam perancangan menggunakan konsep pola *design* arsitektur *MVC* (*Model, Control, View)* Dengan konsep *MVC* ini, seakan memiliki bagian yang terpisah dan bisa dikembangkan masing-masing. Maka, proses perancangan *API* bisa dilakukan lebih cepat karena *developer* akan lebih fokus pada pengerjaan salah satu bagian saja. Berikut Tampilan konsep susunan folder *MVC* pada aplikasi *WeShare* :



Gambar 3. WeShare *Coding Foundation*

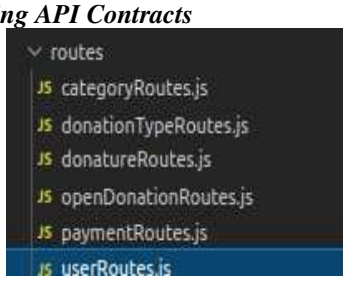


Gambar 4. *WeShare Model*



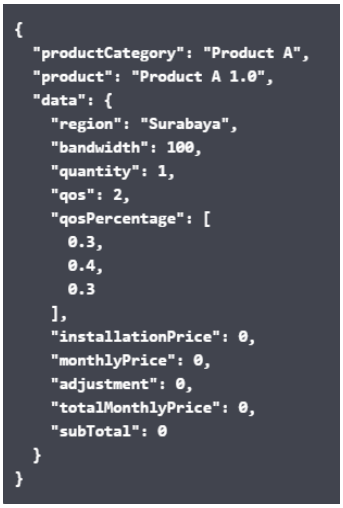
Gambar 5. *WeShare Controllers*

c. Handling API Contracts



Gambar 6. *WeShare Routes*

API Contract adalah lokasi dimana kita mendeklarasikan bagaimana *API* akan berperilaku, termasuk *url* titik akhir, tindakan setiap titik akhir, argumen, contoh *respons*, dan detail lainnya, Hal ini pun terimplementasi dalam pengembangan aplikasi *WeShare*.



```
{
  "productCategory": "Product A",
  "product": "Product A 1.0",
  "data": {
    "region": "Surabaya",
    "bandwidth": 100,
    "quantity": 1,
    "qos": 2,
    "qosPercentage": [
      0.3,
      0.4,
      0.3
    ],
    "installationPrice": 0,
    "monthlyPrice": 0,
    "adjustment": 0,
    "totalMonthlyPrice": 0,
    "subTotal": 0
  }
}
```

Gambar 7. Contoh *API Contract*

Ada beberapa format file yang memungkinkan kita membuat kontrak dan mendapatkan dokumentasi (atau server tiruan, pengujian otomatis, dll.). Yang paling umum saat ini adalah Spesifikasi *Open API* (sebelumnya dikenal sebagai Swagger).

Kontrak ditentukan oleh penyedia layanan dan ditujukan untuk konsumen *API*, dengan kata lain, untuk perusahaan dan pengembang yang akan menggunakan *API*. Dokumen biasanya dibuat oleh tim pengembangan. Jika seorang pengembang memiliki pengetahuan domain yang lengkap, ia dapat membuat dokumentasi sendiri. Kalau tidak, dia mungkin memerlukan bantuan dari beberapa ahli domain.

d. Design Backend Infrastrusture

Sebuah *Infrastructure* dalam *development REST API* harus sangat diperhatikan, karena hal ini

ISSN 2686-3359 (Online)
DOI Prefix 10.51401

menunjang kelancaraan pada saat perancangan, seperti *tools-tools* apa saja yang digunakan. Dalam perancangan *Infrastructure REST API* yang terdapat dalam aplikasi *WeShare* ini terbagi menjadi 3 (tiga) poin utama yakni :

1. Version Control System

Version Control System yang digunakan dalam perancangan *REST API* ini adalah *GIT* dan akses *remote* layanan *repositories* untuk menunjang *version control system* yang digunakan disini adalah *GITLAB*. Kedua *tools* tersebut mempermudah tim untuk kolaborasi hasil kerja serta sebagai perantara layanan untuk proses *push* ke *cloud server* supaya *REST API* dapat di konsumsi oleh *Frontend* dan *React Native*.

Berikut contoh proses *push* menggunakan *GIT*

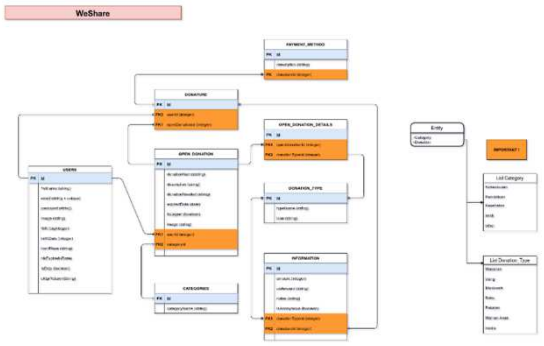
```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)
nothing to commit, working tree clean
hp@LAPTOP-3NKORD70 MINGW64 ~/SampleProject (master)
$ git remote -v
origin https://gitlab.com/binarxglints_batch12/finalproject/team_f/backendteam_f/-/tree/dev_2.1 (fetch)
origin https://gitlab.com/binarxglints_batch12/finalproject/team_f/backendteam_f/-/tree/dev_2.1 (push)

hp@LAPTOP-3NKORD70 MINGW64 ~/SampleProject (master)
$ git push origin master
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 4 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (7/7), 916 bytes | 36.00 KiB/s, done.
Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
To https://gitlab.com/binarxglints_batch12/finalproject/team_f/backendteam_f/-/tree/dev_2.1
```

ke *remote* layanan *repositories GITLAB* :

Untuk akses lengkap repositories REST API WeShare dapat diakses melalui link GITLAB berikut : https://gitlab.com/binarxglints_batch12/finalproject/team_f/backendteam_f/-/tree/dev_2.1

2. Database Model and Database Type

Langkah selanjutnya yang dilakukan dalam perancangan *design infrastructure REST API* ini adalah menentukan model *database* apa yang digunakan, dalam kasus ini model *database* yang digunakan adalah *ORM* (*Object Relation Mapping*), sehingga proses data bersifat berelasi antar tabel satu dengan tabel yang lainnya.



Gambar 8. *WeShare Schema*

Setelah model *database* telah ditentukan maka harus ditentukan pula *type database* yang akan digunakan, berhubung dalam kasus ini model *database* yang digunakan adalah *ORM*, maka type *database* yang digunakan adalah *PostgreSQL* sehingga akan dihasilkan output Skema tabel (*Table Schema*) yang akan memperjelas alur lalu lintas *object* data yang terdapat dalam *database*.

3. Cloud Server

Dalam proses perancangan selanjutnya adalah menentukan *cloud server*, yang mana berfungsi sebagai pusat *hitting REST API* melalui *route* atau *URLs* yang telah dirancang, sehingga dapat dikonsumsi oleh *Frontend* dan *React Native*.

Cloud *server* yang digunakan disini adalah *HEROKU*, karena *cloud server* ini gratis dan mudah dalam konfigurasi. Berikut *link cloud server HEROKU REST API WeShare* : https://*WeShare*-heroku.herokuapp.com/

```
E:\Server Web\bot\meme>git add .

E:\Server Web\bot\meme>git commit -am "make it better"
On branch master
nothing to commit, working tree clean

E:\Server Web\bot\meme>git push heroku master
Counting objects: 16, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (14/14), done.
Writing objects: 100% (16/16), 3.91 KiB | 1.30 MiB/s, done.
Total 16 (delta 5), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> PHP app detected
remote:
remote:  !     WARNING: No 'composer.json' found.
remote:        Using 'index.php' to declare app type as PHP is considered legacy
remote:        functionality and may lead to unexpected behavior.
remote:
remote: -----> Bootstrapping...
remote: -----> Installing platform packages...
remote:        NOTICE: No runtime required in composer.lock; using PHP ^5.5.17
remote:        - apache (2.4.29)
remote:        - nginx (1.8.1)
remote:        - php (5.6.33)
remote: -----> Installing dependencies...
remote:        Composer version 1.6.3 2018-01-31 16:28:17
remote: -----> Preparing runtime environment...
remote:        NOTICE: No Procfile, using 'web: heroku-php-apache2'.
remote: -----> Checking for additional extensions to install...
remote: -----> Discovering process types
remote:        Procfile declares types -> web
remote:
remote: -----> Compressing...
remote:        Done: 13.8M
remote: -----> Launching...
remote:        Released v3
remote:        https://limitless-meadow-28550.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/limitless-meadow-28550.git
 * [new branch]      master -> master
```

Gambar 9. Contoh *Syntax deploy to Heroku*

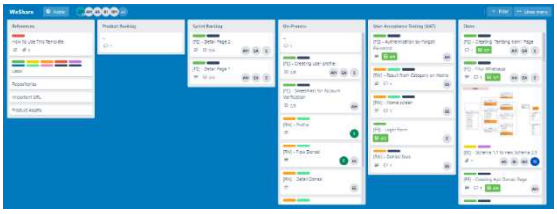
e. Development Methode

Langkah *development* selanjutnya dalam perancangan *WeShare* ini adalah menentukan metode yang digunakan. Dalam proses *development project WeShare*, metode yang digunakan adalah metode *SCRUM*, apa itu metode *SCRUM*?

Metode *SCRUM* merupakan metodologi yang termasuk dalam *agile software development*. Kenapa dalam perancangan aplikasi *WeShare* kami gunakan *SCRUM*, karena *SCRUM* dinilai dapat menghasilkan kualitas perangkat lunak yang baik sesuai dengan keinginan pengguna, dapat digunakan dalam proyek besar maupun kecil, dan mudah untuk mengadopsi perubahan termasuk dalam perancangan aplikasi

ISSN 2686-3359 (Online)
DOI Prefix 10.51401

WeShare ini. Situs yang kami gunakan untuk mengimplementasikan metode ini adalah *TRELLO*.

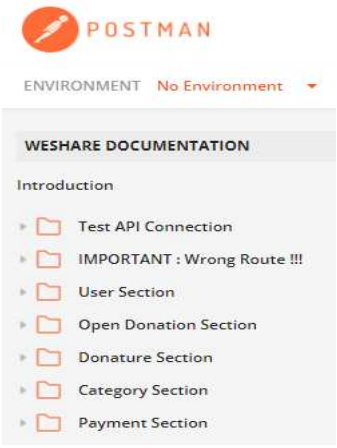


Gambar 10. *SCRUM Process WeShare Development*

Untuk lebih jelasnya *Proses development* menggunakan metode *SCRUM* dalam perancangan aplikasi *WeShare* dapat diakses pada link berikut : https://trello.com/b/VB9eIdSM/weshare

f. *API Release*

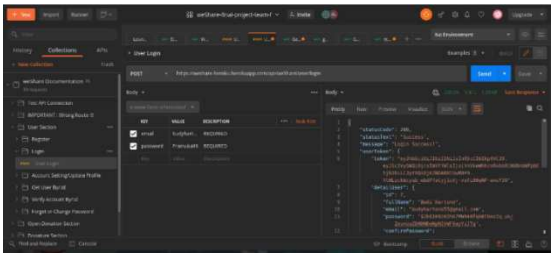
Langkah akhir yakni dilakukan adalah merelease *API* yang telah diselesaikan, akan tetapi sebelum hal itu dilakukan tim *Backend* harus melakukan pengecekan terhadap seluruh *route/URLs* pada setiap *API*, dalam proses ini kami menggunakan *tools POSTMAN* untuk *hitting API*, memastikan bahwa *API* berjalan dengan baik dan sesuai dengan fitur yang diharapkan.



Gambar 11. *WeShare API Section*

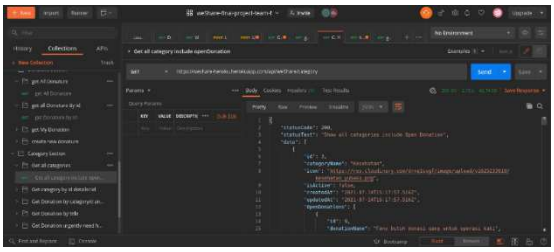
Untuk pereleasan *API* disertakan dengan *API documentation*, yang berguna sebagai panduan tim *Frontend* dan *React Native* untuk *hitting*. WeShare API documentation dapat diakses melalui link Berikut : https://documenter.getpostman.com/view/15855764/TzkyLKh9
Bentuk keluaran *API* yaitu dalam bentuk *JSON*.

Contoh bentuk keluaran *JSON Object* dalam *WeShare API Login* pada saat dilakukan *testing* menggunakan *tools POSTMAN* dapat dilihat pada gambar disamping ini:



Gambar 12. *Hitting API (Login) Menggunakan Postman*

Contoh keluaran *JSON Object* dalam *Category include Open Donation* menggunakan *POSTMAN* :



Gambar 13. Contoh *Output JSON Object Category Menggunakan Postman*

Hasil tampilan data *JSON Object API Category* menggunakan *POSTMAN* diatas sebagai berikut:

Gambar 14. *Output Data JSON Object Category WeShare*

Untuk data *JSON Object Category WeShare* lebih lengkapnya dapat diakses melalui link berikut : https://weshare heroku.herokuapp.com/api/weShare/category
Dan terlampir juga pada *WeShare API Documentation*.

Dalam perancangan REST API untuk aplikasi *WeShare* terdapat beberapa fungsi metode permintaan *response,* ada *endpoint* sebagai alamat *API* guna mempermudah *Hitting API* Tim *developer* terutama yang bekerja sebagai Tim Depan/*Front End* dan terdapat Description sebagai panduan yang digunakan oleh tim untuk menggunakan masing-masing *API.*

Berikut sedikit penjelasan terkait *response* metode yang digunakan dalam *API WeShare*:

ISSN 2686-3359 (Online)
DOI Prefix 10.51401

1. *GET*, dalam penggunaannya *method* untuk kelompok *HTTP verb* ini adalah untuk mengambil atau membaca data. *Method* pada kelompok ini biasanya mengembalikan suatu keluaran/*output* yang kadang bisa disebut sebagai *function*.
2. *POST*, dalam penggunaannya *method* untuk kelompok *HTTP verb* ini adalah untuk membuat (*create*) *item*/*resource* baru. Kelompok *method* ini biasanya tidak mengembalikan keluaran/*output* yang kadang disebut *procedure*.
3. *PUT*, dalam penggunaannya *method* untuk kelompok *HTTP verb* ini adalah untuk mengupdate *item*/*resource* yang telah ada. (sama dengan *point* 2).
4. *DELETE*, dalam penggunaannya method untuk kelompok *HTTP verb* ini adalah untuk menghapus *item*/*resource* yang telah ada. (sama dengan *point* 2).

3. Perancangan *API*

Routes	EndPoint	Description
POST	/api/weShare/user/register	Register user
POST	/api/weShare/user/login	login user to get token for authentication
GET	/api/weShare/user/:id	API get User by/id
PUT	/api/weShare/user/forgetPassword	API for changed Password
PUT	/api/weShare/user/verifyAccount/:id	API for verify Account (by /id)
PUT	/api/weShare/user/editProfile/:id	API for edit user profile (by /id)
GET	/api/weShare/allCategory	API for get all category include Opendonation
GET	/api/weShare/category/details/:id	API for get Opendonation by category id
POST	/api/weShare/addCategory	API for create category (developer only)
PUT	/api/weShare/editCategory	API for edit category by id (developer only)
POST	/api/weShare/createDonature	create new donature
GET	/api/weShare/allDonature	get all donature
GET	/api/weShare/donatureById/:id	get donature by id
GET	/api/weShare/myDonation?UserId=1&Type=1	get my donation by UserId and DonationTypeId
POST	/api/weShare/openDonation/create	create new Open Donation
GET	/api/weShare/OpenDonation	get all Open Donation
GET	/api/weShare/OpenDonation/:idOpenDonation	detail Open Donation
PUT	/api/weShare/OpenDonation/update/:idOpenDonation	update Open Donation
DELETE	/api/weShare/OpenDonation/delete/:idOpenDonation	delete Open Donation
GET	/api/weShare/OpenDonation/my	Get openDonation by id user
GET	/api/weShare/category/urgent	Get openDonation need donation urgently
GET	/api/weShare/category/newest	Get Newest openDonation
GET	/api/weShare/category/donationTitle	Get openDonation by title
GET	/api/weShare/category/donation	Get Opendonation by category id and donationType id
POST	/api/weShare/createPayment	create payment
GET	/api/weShare/getPayment	Get all payment
PUT	/api/weShare/updatePayment/:id	update payment by id
DELETE	/api/weShare/createPayment/:id	Delete payment by id
GET	/api/weShare/allCategory	Get All Category
GET	/api/weShare/allDonationType	Get All Donation Type

Last Update on (19 July 2021 | 22:00 33)

BACKEND DEVELOPER (TEAM F):

```
$ Budi Hartono     (BackEnd Leader)
$ Nandra           (BackEnd Co-Leader)
$ Ari seno         (Developer)
$ Beni Iskandar    (Developer)
```

How to run

Server

```
$ cd server
$ npm i
$ npm start
```

Client

```
$ cd client
$ npm i
$ npm start
```

KESIMPULAN DAN SARAN

Dari hasil penelitian ini diperoleh suatu kesimpulan bahwa telah dapat dirancang bangun sistem berbasis *REST API* dengan arsitektur *REST* dari sisi *Backend Development* untuk diimplementasikan pada client Website dan Mobile *(WeShare)* terkait transaksi pelayanan donasi. Sistem yang dirancang untuk pelayanan donasi online *(WeShare App)* ini menggunakan konfirmasi donasi melalui aplikasi yang didalamnya terdapat form yang harus diisi terkait *donasi user* kemudian *upload* foto struk sebagai bukti pembayaran. Konfirmasi pembayaran yang dilakukan *user* melalui *contact Whatsapp* sehingga akan dilakukan *check* oleh *admin* melalui *server database*, jika data dan bukti pembayaran terverifikasi maka *admin* akan melakukan update status *User Transaction* dari *unpaid* menjadi *paid*, selanjutnya detail pembayaran *user* akan dikirim melalui fitur *live chat WhatsApp* oleh admin.

Saran kami terkait hasil penelitian ini adalah sistem ini dapat dikembangkan menjadi suatu portal terintegrasi dengan berbagai aplikasi di masyarakat, misalnya aplikasi marketplace, aplikasi e-walet dan lain sebagainya.

DAFTAR PUSTAKA

[1] A. Kadir. (2006). "*From Zero To A Pro* Pemrograman Aplikasi Android", p. 408, 2014, doi: 10.13140/2.1.1589.0563.

[2] Irfan Kurniawan, Humaira, & Fazrol Rozi. (2020). *REST API* Menggunakan NodeJS Pada Aplikasi Transaksi Jasa Elektronik Berbasis Android.

[3] M. Akbar. (2018). Pengembangan *Restful Api* Untuk *Application Specific High Level Location Service.*

[4] Sari Noorlima Yanti, Erni Rihyanti. (2021). Penerapan *Rest API* Untuk Sistem Informasi Film Secara Daring.