

LAPORAN TUGAS BESAR
SISTEM PEMBAYARAN (*PAYMENT SYSTEM*)

KELOMPOK: MATA DUITAN

Diajukan untuk memenuhi syarat kelulusan mata kuliah

EL4236 Perancangan Perangkat Lunak Jaringan



Disusun oleh:

Rizky Ardi Maulana 13217054

Moh. Tamamul Firdaus 13217062

Arief Hirmanto 13217076

PROGRAM STUDI TEKNIK ELEKTRO
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2021

Latar Belakang dan Deskripsi



Digital payment merupakan suatu metode pembayaran elektronik menggunakan sms atau layanan internet. Sistem ini memudahkan pengguna melakukan transaksi tanpa perlu kontak fisik langsung dengan uang atau lawan transaksi. Salah satu *digital payment* yang sedang berkembang adalah *e-wallet* seperti Paypal, Gopay, Link-Aja, dan Ovo. *E-wallet* menggunakan internet sebagai jalur komunikasi. *E-wallet* memungkinkan pengguna untuk mengirim uang atau membayar tagihan tanpa mengharuskan memiliki rekening di bank. Untuk tugas besar mata kuliah Perancangan Perangkat Lunak Jaringan ini, kami terinspirasi untuk membuat perangkat lunak serupa. Perangkat lunak yang kami rancang bernama “Mata Duitan” dan merupakan versi yang lebih sederhana dari *e-wallet* yang sudah ada. Sistem yang dirancang akan menggunakan *server* sebagai pengolah data pembayaran. Lalu pengguna (*user*) yang ingin melakukan transaksi akan dianggap sebagai *client*.

Deskripsi Fungsionalitas

1. Mengirim Uang (Transfer)

Mengirim uang adalah salah satu fitur utama dari aplikasi yang dirancang. Fitur ini menampilkan sebuah formulir dengan rincian sebagai berikut:

- Recipient : username dari akun penerima uang
- Amount : jumlah uang yang dikirim
- Description : Deskripsi atau keterangan yang dapat ditambahkan pengguna terkait pengiriman uang

2. Membuat Tagihan (Create Bill)

Membuat Tagihan adalah fitur yang kami tujukan untuk membuat tagihan utang kepada penerimanya. Fitur ini menampilkan sebuah formulir pada antarmuka dengan rincian sebagai berikut:

- Bill Name : Nama tagihan, cth : Pembelian *Valorant Point* (Koin dalam game valorant)
- Bill Recipient : username dari akun penerima tagihan
- Amount : jumlah uang yang ditagih
- Description : Keterangan/deskripsi tambahan dari pengguna terkait tagihan (cth : tolong bayar segera)

Ketika tagihan dibuat, sebuah kode tagihan dibuat secara otomatis untuk digunakan sebagai penanda tagihan. Kode tagihan tersebut akan digunakan saat dilakukan proses pembayaran tagihan.

3. Membayar Tagihan (Pay Bill)

Membayar Tagihan adalah fitur untuk membayar tagihan-tagihan yang telah ditujukan kepada pengguna. Fitur ini menampilkan daftar tagihan yang ditujukan pengguna. Tiap tagihan memiliki kode yang unik. Untuk melakukan pembayaran tagihan, pengguna perlu memasukkan kode tagihan yang ditampilkan pada daftar tagihan. kode tagihan akan merujuk tagihan yang terdapat pada daftar tagihan. Dengan mengisi kode tagihan, pengguna tidak perlu lagi mengisi formulir transfer secara manual. Dengan demikian, proses pembayaran tagihan dapat lebih efisien.

Spesifikasi Teknis

A. Physical Network, Network Addressing, Transport Layer

- Physical Network : WAN/Internet
- Network Addressing : IP Address version 4 (IPv4)
- Transport Layer : TCP
- Application Layer : HTTP

B. Client-Server Option

Perangkat		Resource yang digunakan
Client	Gui	Python Tkinter
	Client API	Python
Server		Express (Node JS)
Database		MongoDB

C. Security

Untuk pengamanan API, digunakan sistem JWT (JSON Web Token). Sistem JWT ini digunakan untuk memproteksi API supaya penggunaan/pengaksesan API dibatasi hanya ke pengguna saja. Untuk memperoleh token, user nantinya akan melakukan login untuk kemudian diperoleh token yang bisa digunakan user. Token ini berlaku selama 1 hari dan dipasang di *header* setiap HTTP Request kepada API. Untuk pengaksesan pengguna sendiri terdapat autentikasi standar berupa login dengan username dan password. Bila password/username salah, maka token tidak diberikan dan sistem tidak bisa diakses.

Untuk keamanan proses transaksi, diimplementasikan *One-Time Password* (OTP) khusus untuk transaksi. Apabila OTP salah, maka transaksi tidak bisa diproses. Dalam melakukan transaksi, diperlukan token (untuk mengakses API) dan OTP (sebagai verifikasi) sebelum transaksi dikatakan valid dan berhasil.

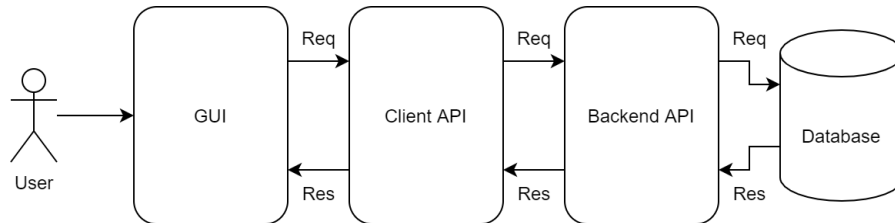
D. Typical Message

Message akan dikirim/diproses dalam format JSON dan menggunakan HTTP request berupa POST dan GET.

- Client-to-server
 - ID user
 - Password
 - Amount
 - Description
 - PIN/OTP
- Server-to-client
 - ID user
 - Amount
 - Description
 - Timestamp

Perancangan Sistem

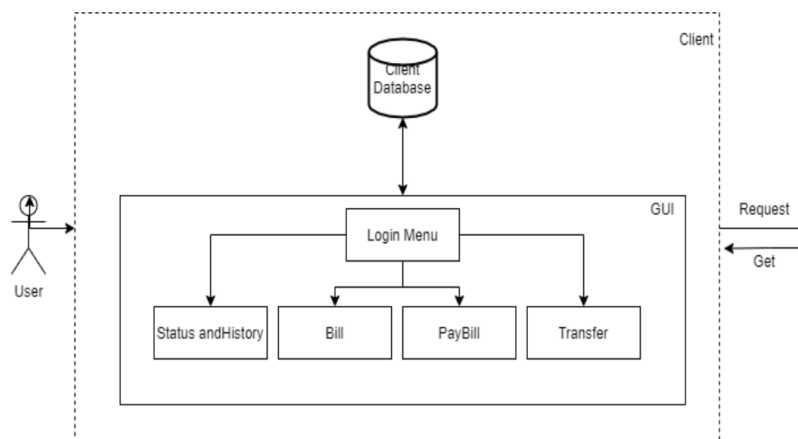
Dalam perancangan sistem, sistem ini terbagi menjadi 4 buah komponen. Komponen digambarkan dalam diagram berikut:



Komponen-komponen sistem yaitu *Graphical User Interface (GUI)*, *Client API* (API untuk pemrosesan data *client*), *Backend API* (API untuk berkomunikasi dengan *database*), *database* (untuk menyimpan data).

Graphical User Interface (GUI)

Arsitektur dari *Graphical User Interface (GUI)* adalah sebagai berikut:



Pada GUI, User akan melakukan proses login/sign-in lalu kemudian memasuki halaman utama / *home*. Pada halaman *home* nanti akan tertera informasi user beserta jumlah uang yang dimiliki user. Dari halaman *home* tersebut, user dapat mengakses beberapa menu yang dijelaskan sebagai berikut:

- Menu Pay Bill

Menu ini digunakan untuk melihat *bill* / tagihan apa saja yang dimiliki user. User kemudian dapat memilih tagihan apa yang akan dibayar untuk kemudian akan diproses dengan mengambil kredit/*balance* dari user dan menghapus tagihan user. Saat proses pembayaran, user perlu memasukkan OTP yang nanti diberikan oleh Backend API.

- Menu Transfer

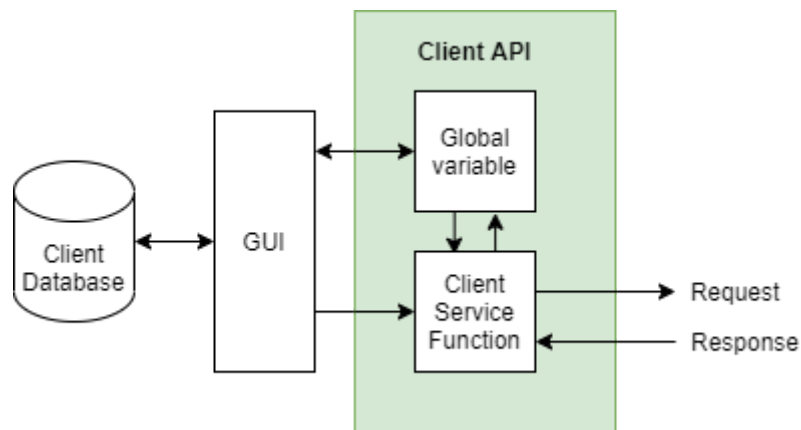
Menu digunakan oleh user untuk melakukan transfer kepada user lain. Kredit dari user kemudian akan dikurangi dan ditambahkan kepada user tujuan. Sama seperti Pay Bill, user perlu memasukkan kode OTP yang nanti diberikan oleh Backend API

- Menu Bill

Menu ini digunakan oleh user untuk membuat tagihan kepada user lain. Tagihan ini kemudian akan tampil pada menu Pay Bill user lain. Sama seperti Pay Bill, user perlu memasukkan kode OTP yang nanti diberikan oleh Backend API.

Client API

Untuk arsitektur dari Client API ditampilkan sebagai berikut:



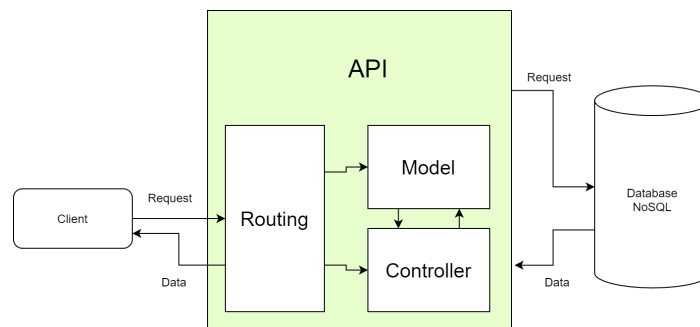
Untuk Client API, komponen ini akan terhubung dengan Backend API untuk memproses data dari Backend API serta mengirim HTTP Request ke Backend API. Data yang diambil dari Backend API akan disimpan ke dalam Global Variable yang bisa diakses oleh GUI. Beberapa *services* yang dijalankan oleh Client API ini antara lain:

- *Services* Registrasi akun
- *Services* autentikasi login (setelah berhasil, akan menyimpan data user + token)
- *Services* transfer uang
- *Services* menambah tagihan
- *Services* membayar tagihan
- *Services* memverifikasi user tujuan (untuk tagihan dan transfer)
- *Services* OTP (meminta OTP)

Client API diimplementasikan dengan Python dan *library* requests untuk melakukan HTTP Request kepada Backend API. Pada *request*, diselipkan token pada *header* untuk verifikasi akses Backend API.

Backend API

Untuk arsitektur Backend API beserta *database*, ditampilkan sebagai berikut:



Backend API ini dibuat memanfaatkan arsitektur *Model-View-Controller* (untuk View tidak dibuat) dan berbasis REST API. Backend API ini berguna untuk menerima HTTP Request dari Client API untuk diproses dan melakukan tindakan sesuai *request*. Sub komponen Model disini berguna untuk menyimpan/mengambil data pada database sesuai dengan skema/tabel yang didefinisikan pada Model. Sub komponen Controller berguna untuk mengatur *logic* atau aliran pemrosesan data dari database. Untuk *interface* dari Client API, digunakan Routing sebagai URL untuk menerima *request* dari Client API. Saat menerima *request*, Backend API akan memverifikasi token JWT dari Client API untuk kemudian melanjutkan akses *request*.

Beberapa *services* yang diterapkan pada Backend API antara lain:

- *Services* autentikasi login
- *Services* transfer uang
- *Services* menambah tagihan
- *Services* menghapus tagihan
- *Services* menyimpan data transaksi
- *Services* generate dan verifikasi OTP

Perbedaan *services* ini dengan Client API adalah pada Backend API, *services* terhubung langsung dengan data dari database dimana pada Client API *logic*-nya terhubung dengan user secara langsung.

Untuk Routing, URL dari Backend API adalah sebagai berikut (setiap *route* membutuhkan token JWT):

- *routing* untuk autentikasi:
 - `/api/auth/signin` (HTTP POST, *body username, password*) => Melakukan proses login dan menerima data user beserta token yang valid.
 - `/api/auth/signup` (HTTP POST) => Melakukan proses registrasi user baru

- *routing* untuk tagihan/bill:
 - /api/bill/ (HTTP GET) => Mendapatkan semua tagihan dari semua user
 - /api/bill/ (HTTP POST) => Membuat tagihan
 - /api/bill/<bill_owner> (HTTP GET) => Mendapatkan tagihan berdasarkan *username* pemilik tagihan
 - /api/bill/<bill_id> (HTTP DELETE) => Menghapus tagihan
- *routing* untuk User
 - /api/user/ (HTTP GET, *body* username) => Mendapatkan data user tertentu berdasarkan *username* di *body request*
 - /api/user/transfer (HTTP PUT, *body* username, credit, category) => Mengubah kredit/uang dari user tertentu
 - /api/user/bill (HTTP PUT, *body* username, bill_id) => Menambahkan tagihan pada user
 - /api/user/transfer/<username> (HTTP GET) => Mencari riwayat transfer/bayar tagihan berdasarkan *username*
- *routing* untuk generate OTP:
 - /api/processing (HTTP GET) => *Generate* OTP dan menyimpan dalam global variabel.

Untuk verifikasi OTP sendiri, *request* dari Client API yang memiliki *body* OTP akan diverifikasi oleh Backend API sebelum proses transaksi berhasil.

Database

Untuk *database* sendiri, terdapat beberapa tabel/koleksi sebagai berikut:

- User, terdiri dari:
 - Username
 - Password (di-hash)
 - Nama
 - Kontak
 - Email
 - Bill ID (tagihan-tagihan user)
 - Timestamp
- Bill, terdiri dari:
 - Bill Name (nama tagihan)

- Bill ID (ID tagihan)
- Bill Owner (pemilik tagihan)
- Bill Sender (pengirim tagihan)
- Amount (jumlah uang)
- Description (deskripsi tagihan)
- Timestamp
- Transfer, terdiri dari:
 - Username
 - Change Balance (perubahan uang dari user)
 - Category (apakah perubahan uang ini karena transfer atau pembayaran tagihan)
 - Timestamp

Pada tabel/koleksi Transfer berfungsi sebagai *track*/riwayat uang yang keluar atau masuk dari user dengan kategori pembayaran tagihan ataupun transfer.

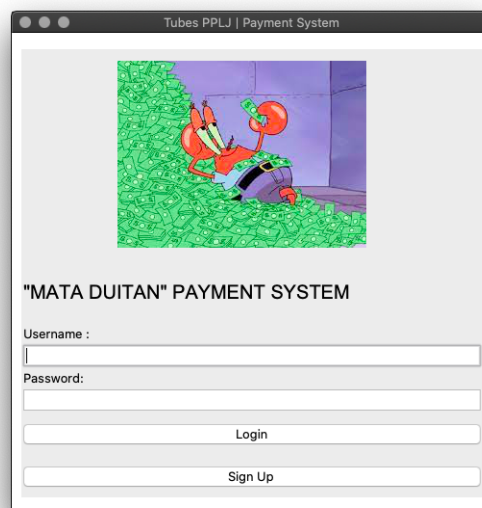
Implementasi

Graphical User Interface (GUI)

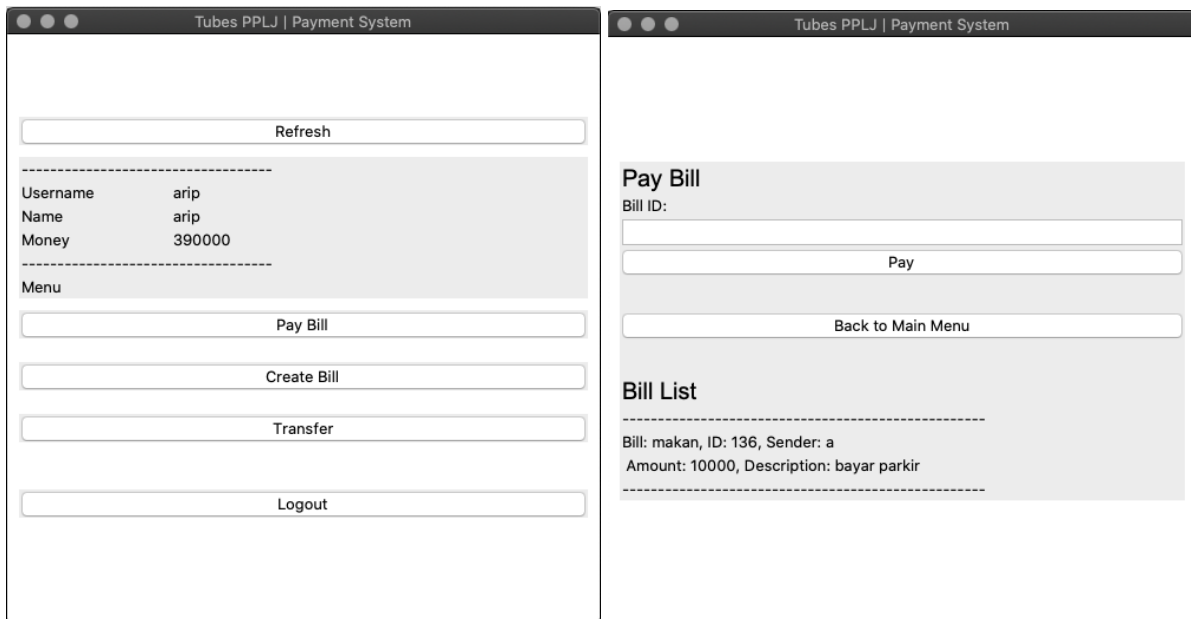
Implementasi GUI dilakukan dengan menggunakan library **Tkinter** pada python. Berikut salah satu potongan kode yang digunakan untuk membuat halaman login.

```
root = tk.Tk()
root.title("Tubes PPLJ | Payment System")
root.geometry("500x500")
login=Login(root)
root.mainloop()
```

Hasil yang ditampilkan yaitu sebagai berikut.



Selanjutnya dibuat juga halaman untuk menu, dan juga halaman setiap sub-menu. Berikut potongan gambarnya.



Fungsi `tkk.Label` digunakan untuk menampilkan text ke dalam window sedangkan button diimplementasikan dengan fungsi `tkk.Button`. Berikut potongan kode yang digunakan untuk membuat tampilan Label dan Button.

```
# menu
tkk.Label(self.frame, text='Menu').pack(fill='x', expand=True)

# Pay Bill button
tkk.Button(self.frame, text='Pay Bill',
            command=self.make_page_1).pack(fill='x', expand=True, pady=10)
self.page_1 = Page_1(master=self.root, app=self)
```

Selanjutnya saat tombol ditekan akan dipanggil fungsi yang berfungsi untuk memproses langkah selanjutnya seperti pindah halaman atau melakukan request ke server. Berikut contohnya.

```
def login_clicked(self):

    flag = Signin(self.username.get(), self.password.get())

    if(flag==0):
        self.signin.pack_forget()
        app = App(root)
    elif(flag==1):
        showinfo(
            title='Information',
            message="User not Found")
```

```

    )
    elif(flag==2):
        showinfo(
            title='Information',
            message="Wrong Password"
        )
    # print(flag)

```

Fungsi **def_login** akan dipanggil saat tombol login ditekan. Fungsi **def_login** akan melakukan request ke server dengan cara memanggil service/fungsi api client yg telah dibuat `flag = Signin(self.username.get(), self.password.get())`. Selanjutnya **def_login** juga akan membuat window yang baru apabila proses login berhasil. Kode yang digunakan yaitu `self.signin.pack_forget()` untuk menutup window tampilan login dan `app = App(root)` untuk membuat window tampilan menu.

Client API

Client API diimplementasikan menggunakan bahasa pemrograman python. Untuk melakukan komunikasi dengan server menggunakan HTTP, digunakan module/library **request**. Terdapat beberapa *service* yang telah dirancang untuk menjalankan aplikasi dari sisi *client*.

- *Services* Registrasi akun

Untuk melakukan registrasi akun, parameter-parameter yang telah diisi di GUI diteruskan ke fungsi SignUp(form). Fungsi ini akan melakukan post ke server untuk menambahkan informasi dari *user* baru ke database server.

```

# Sign Up API
# This api will post user registration to server and the server will
save user registration in database
# Param :
# - Form : signup form in json format.
def SignUp(form) :
    url = var._signup_URL
    # Post form to server
    requests.post(url, json = form)
    # print(form)

```

- *Services* autentikasi login (setelah berhasil, akan menyimpan data user + token)

Untuk menjalankan autentikasi login, fungsi SignIn() dipanggil dengan parameter username *user* dan passwordnya. fungsi ini akan menghasilkan keluaran berupa flag untuk memberi tanda hasil autentifikasi :

- 0 : Berhasil
- 1 : Username tidak ditemukan di server database
- 2 : Username ditemukan namun password salah

Apabila proses autentifikasi berhasil, fungsi akan menyimpan informasi dari pengguna dan juga token pada variable global.

```
# Sign In API
# Param :
# - _username : username of user account
# - _password : _password of user account
# return :
# - status : -> 0 : if user and password match
#           -> 1 : if user is not found in server database
#           -> 2 : if user is found but password doesn't match

def Signin(_username, _password) :
    form = {
        "username" : _username,
        "password" : _password
    }
    status = 0

    url = var._signin_URL
    response = requests.post(url, json = form)
    if (response.status_code == 404): #User not found
        status = 1
    elif (response.status_code == 401) : #Wrong Password
        status = 2
    else :
        status = 0
        #saving var.token session
        print("Posting to " + url)
        buffer = response.json()
        var.token = buffer['accessToken']
        var.username = buffer['username']
        var.name = buffer['name']
        var.amount_credit = buffer['amount_credit']
        print(var.token )
```

```

    # print(var.username)
    print(buffer)
    return status

```

- *Services* transfer uang

Services transfer uang diimplementasikan dengan memanggil fungsi `TransferMoney()`. Fungsi ini menerima parameter berupa username penerima, jumlah uang yang ditransfer, dan deskripsi transfer, dan kategori pengiriman uang dari GUI. Untuk parameter OTP dan token diteruskan dari variable global. Untuk jenis *services* ini parameter kategori diisi “transfer”. Apabila proses transfer uang berhasil akan diberi flag 0 sedangkan gagal diberi flag 1.

```

# Transfer Money API
# This API performs transfer money activity
# param :
# - _recipient      : username of recipient's account
# - _amount_       : amount of money going to be transferred
(string)
# - _description   : description of transfer
# - _OTP           : OTP
# - token          : token session
# _category        : "transfer" or "paybill"
# return :
# success -> 0 : Transfer money success
#          -> 1 : Transfer money failed

def TransferMoney(_recipient, _amount_, _description, _OTP, token,
                 _category):
    credit_url = var._user_URL + '/transfer'
    success = 0
    #Change Recipient Credit
    success = ChangeCredit(credit_url, _recipient,
(int(_amount_)), _OTP, token, _category)
    #Change Sender Credit
    success = ChangeCredit(credit_url, var.username,
(-1)*(int(_amount_)), _OTP, token, _category)
    return success
    # print("Transferring Money Rp"+str(_amount_)+ " to "+_recipient)

```

- *Services* menambah tagihan

Untuk mengimplementasikan *services* ini dipanggil fungsi `CreateBill`. Fungsi ini menerima parameter yang diteruskan dari GUI dan OTP yang dari variable global. Pada fungsi ini, token tidak dijadikan sebagai parameter karena langsung dipanggil dari dalam fungsi. Pada fungsi ini juga dibuat suatu kode tagihan (`bill_id`) sebagai penanda tagihan. Parameter-parameter yang dimasukkan dan kode tagihan kemudian dibentuk ke dalam format json. Pada fungsi ini, dilakukan dua kali proses yakni Post untuk menambahkan bill ke koleksi seluruh bill di database server dan Put untuk memperbarui bill milik penerima bill.

```
# Create Bill API
# This API will create a bill form and post it to server
# Param :
# - _bill_name      : name of the bill
# - _recipient      : username of recipient's account
# - _description    : description of the bill
# - _OTP            : OTP
# Return :
# Success -> 1 : Bill creation failed due to wrong OTP
#           -> 0 : Bill creation is success
#
def CreateBill(_bill_name, _recipient, _amount, _description, _OTP):
    bill_id = str(randint(0,1000))
    form = {
        "bill_name"      : _bill_name,
        "bill_id"        : bill_id,
        "bill_sender"    : var.username,
        "bill_owner"     : _recipient,
        "amount"         : _amount,
        "description"    : _description,
        "otp"            : _OTP
    }
    form_2 = {
        "bill_id": bill_id,
        "username" : _recipient,
        "otp"      : _OTP
    }
    #Post to server
    response = requests.post(var._bill_URL, json=form, headers =
{'X-Access-Token': var.token})
    print('Creating bill to ' + var._bill_URL)
    response = requests.put(var._user_URL+'/bill', json=form_2,
```



```

headers = {'X-Access-Token': var.token})

    if (response.status_code == 401): #wrong otp
        success = 1
    else:
        success = 0

    return success

```

- *Services* membayar tagihan

Implementasi *Services* membayar tagihan mirip dengan *Service* membayar tagihan. Kedua *services* ini menggunakan fungsi `TransferMoney()` untuk mengirim uang. Perbedaan mendasar pada fungsi ini adalah pengguna tidak perlu mengisi formulir transfer pada GUI melainkan cukup memasukkan Kode tagihan atau Bill Id. Setelah memasukkan kode tagihan, parameter-parameter yang terdapat pada fungsi `TransferMoney()` secara otomatis akan terisi. Untuk mencari data terkait tagihan, digunakan fungsi `FindBill()`. Fungsi ini akan mencari apakah kode tagihan ada pada koleksi tagihan pengguna. Apabila ditemukan maka fungsi ini akan mengembalikan indeks tagihan dari pengguna.

```

# Find Bill API
# This fuction is used to find user's bill by matching the bill id
# Param :
# - _bill_id : bill id of bill
# return :
# - found : True    -> bill id is found in user's bill collection
#           False   -> bill id is not found in user's bill collection
# - i      : array index of user's bill collection
def FindBill(_bill_id):
    found = False
    for i in range(0, len(var.bill_form)):
        if (var.bill_form[i]['bill_id'] == _bill_id) :
            found = True
            break

    return (found, i)

```

- *Services* memverifikasi user tujuan (untuk tagihan dan transfer)

Verifikasi user tujuan digunakan untuk proses validasi. verifikasi user tujuan diimplementasikan dengan memanggil fungsi `CheckRecipient()`. Fungsi ini memilik

parameter berupa username dari user yang dicari. Pada fungsi ini, client melakukan request berupa get untuk memperoleh informasi mengenai user yang dicari. Apabila user ditemukan di server database maka akan memberikan keluaran 0 sedangkan apabila user tidak ada maka akan memberikan keluaran 1.

```
# Check Recipient API
# This API checks if the recipient's username account does exist in
server database
# Param :
# - _recipient : username of recipient's account
# return :
# 1 -> recipient not found
# 0 -> recipient exists
def CheckRecipient(_recipient):
    #Check if recipient does exist
    # Find recipient in database
    response = requests.get(var._user_URL, json =
{'username':_recipient}, headers = {'X-Access-Token': var.token})
    if(response.status_code == 404):
        #User not found
        print(_recipient+ ' Not Found')
        return 1
    else :
        return 0
```

- *Services* OTP (meminta OTP)

Services ini meminta server untuk membuat kode OTP yang unik. *Services* ini diimplementasikan dengan fungsi RequestOTP(). Fungsi ini melakukan request berupa get ke server. Apabila server telah menerima permintaan, server mengirimkan pesan balik yang menyatakan kode OTP telah dibuat.

```
# Request OTP API
# This API ask server to generate OTP
def RequestOTP():
    response = requests.get(var._otp_URL, headers =
{'X-Access-Token': var.token})
    print(response.json())
```

- Akses Database Client

Setiap client akan memiliki sebuah collection yang diletakkan dalam database client. Setiap collection akan diberi nama **log_<username>**. Data yang disimpan merupakan data *history* transaksi yang telah dilakukan. Berikut kode yang digunakan.

```
from pymongo import MongoClient
from datetime import datetime

# saving data history
client = MongoClient('mongodb://localhost:27017/')
db = client.mata_uitan_clients
```

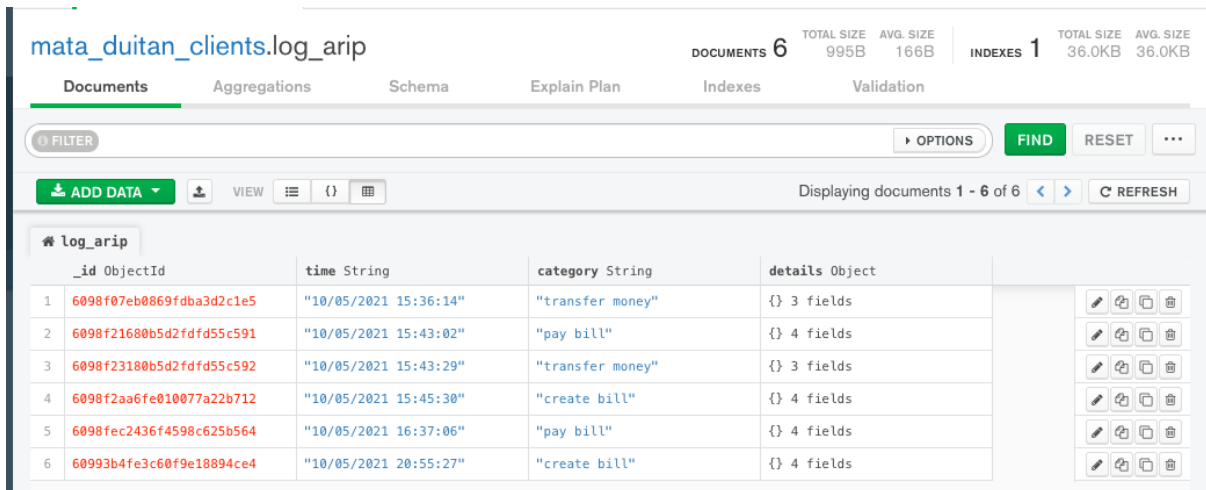
Library yang digunakan untuk menyimpan database yaitu pymongo. Hal ini dikarenakan database client diimplementasikan dengan mongoDB. Akses database diawali dengan membuat koneksi ke database dengan url 'mongodb://localhost:27017/'. Selanjutnya akan dipilih nama database yang digunakan yaitu `mata_uitan_clients`.

Selanjutnya setiap kali transaksi berhasil akan dilakukan penyimpanan data ke database dalam bentuk JSON. Berikut salah satu kode yang digunakan untuk menyimpan data ke collection di database.

```
# save log
data_log = {"time" : datetime.now().strftime("%d/%m/%Y %H:%M:%S"),
            "category" : "transfer money",
            "details" : {
                "recipient" :
self.app.page_3.transfer_recipient.get(),
                "amount" : self.app.page_3.amount.get(),
                "description": self.app.page_3.description.get()
            }
}
print(data_log)
col_name = "log_" + var.username
db[col_name].insert_one(data_log)
```

Penyimpanan data *history* transaksi ini dapat digunakan untuk mengamati seluruh transaksi yang telah dilakukan oleh sebuah client. Data log/*history* disimpan dengan menambahkan waktu transaksi dengan memanfaatkan fungsi `datetime.now().strftime("%d/%m/%Y %H:%M:%S")`.

Nantinya data transaksi akan terlihat seperti berikut di database.



	_id ObjectId	time String	category String	details Object
1	6098f07eb0869fdb3d2c1e5	"10/05/2021 15:36:14"	"transfer money"	{} 3 fields
2	6098f21680b5d2fd55c591	"10/05/2021 15:43:02"	"pay bill"	{} 4 fields
3	6098f23180b5d2fd55c592	"10/05/2021 15:43:29"	"transfer money"	{} 3 fields
4	6098f2aa6fe010077a22b712	"10/05/2021 15:45:30"	"create bill"	{} 4 fields
5	6098fec2436f4598c625b564	"10/05/2021 16:37:06"	"pay bill"	{} 4 fields
6	60993b4fe3c60f9e18894ce4	"10/05/2021 20:55:27"	"create bill"	{} 4 fields

Backend API

Pada backend API, implementasi dibuat dengan Express (Node JS). Dengan arsitektur MVC, maka dibuat model Bill, User, Role (untuk pengembangan), dan Transfer. Untuk *source code* dari API dapat diamati pada *file* terpisah yang disertakan bersamaan dengan laporan ini. Berikut adalah *project tree directory* untuk tugas ini:

```
├── api
│   └── app
│       ├── config
│       │   ├── auth.config.js
│       │   └── db.config.js
│       ├── controllers
│       │   ├── auth.controller.js
│       │   ├── bill.controller.js
│       │   ├── otp.controller.js
│       │   └── user.controller.js
│       ├── middlewares
│       │   ├── authJwt.js
│       │   ├── index.js
│       │   ├── serviceOTP.js
│       │   ├── verifySignUp.js
│       │   └── verifyUsername.js
│       ├── models
│       │   ├── bill.model.js
│       │   ├── index.js
│       │   ├── role.model.js
│       │   ├── transfer.model.js
│       │   └── user.model.js
│       ├── routes
│       │   ├── auth.routes.js
│       │   ├── bill.routes.js
│       │   ├── otp.routes.js
│       │   └── user.routes.js
│       ├── node_modules
│       ├── OTPServer.js
│       ├── package-lock.json
│       └── package.json
```

Untuk pengujian Backend API digunakan aplikasi Postman:

- Proses Login


```
1 {
2   "bill_name": "Tagihan #1",
3   "bill_id": "123",
4   "bill_owner": "akuprabu",
5   "bill_sender": "a",
6   "amount": 500,
7   "description": "transfer",
8   "otp": "6762"
9 }
```

```
1 {
2   "_id": "6099411973d6b917e40b2f06",
3   "bill_name": "Tagihan #1",
4   "bill_id": "123",
5   "bill_owner": "akuprabu",
6   "bill_sender": "a",
7   "amount": 500,
8   "description": "transfer",
9   "isPaid": false,
10  "createdAt": "2021-05-10T14:28:09.611Z",
11 }
```

Terlihat tagihan sudah berhasil dibuat. Apabila kode OTP salah:

```
1 {
2   "bill_name": "Tagihan #1",
3   "bill_id": "123",
4   "bill_owner": "akuprabu",
5   "bill_sender": "a",
6   "amount": 500,
7   "description": "transfer",
8   "otp": "8988"
9 }
```

```
1 {
2   "message": "Klrong OTP"
3 }
```

Transaksi tidak diproses lebih lanjut

- Proses menghapus tagihan

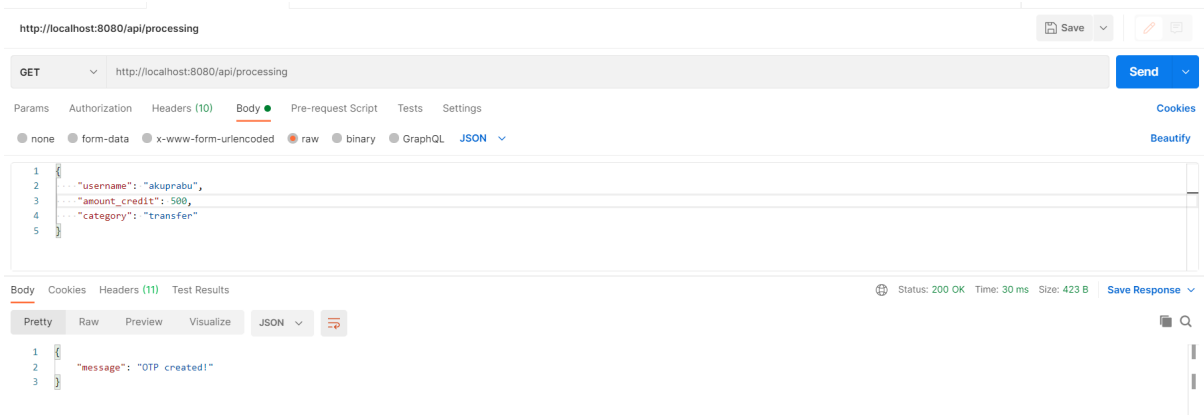
```
1 {
2   "bill_name": "Tagihan #1",
3   "bill_id": "112",
4   "bill_owner": "akuprabu",
5   "bill_sender": "a",
6   "amount": 1000,
7   "description": "Silahkan bayar tagihan"
8 }
```

```
1 {
2   "message": "Bill was deleted successfully!"
3 }
```

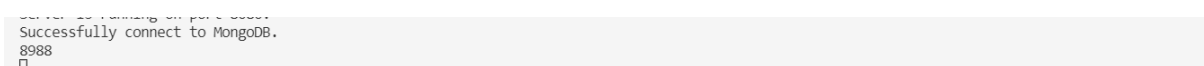
Tagihan sudah berhasil dihapus

- Proses melakukan perubahan kredit user

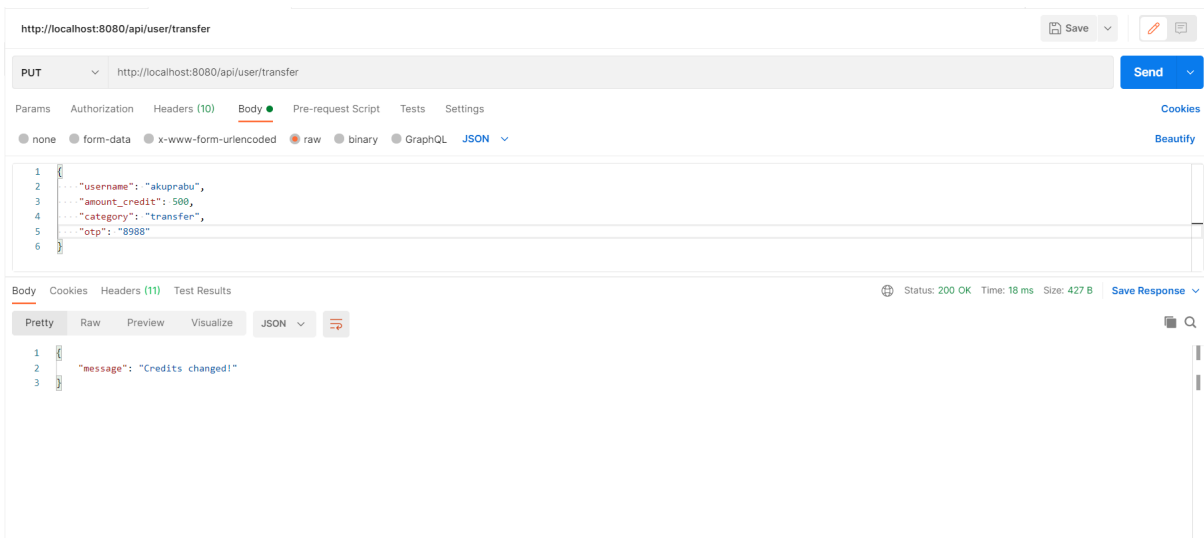
Pertama minta OTP terlebih dahulu



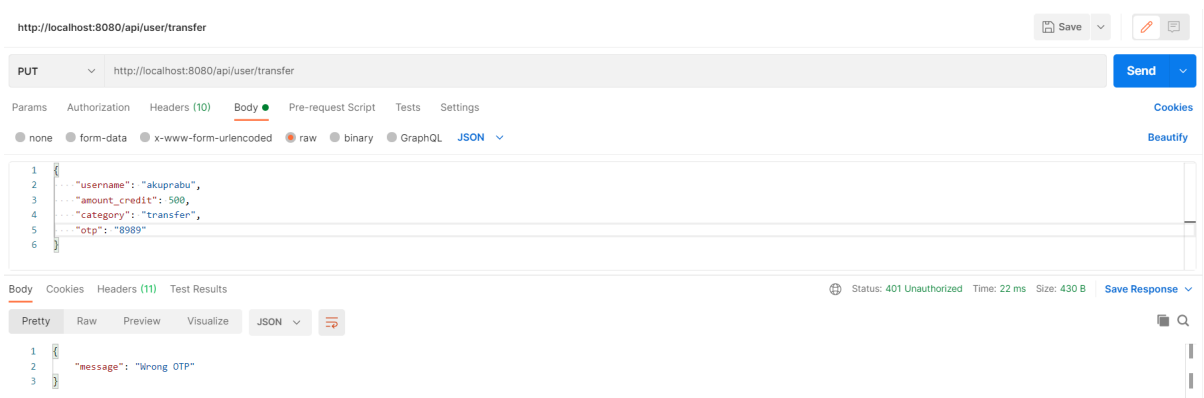
OTP akan di-*print* pada terminal server:



Kemudian lakukan *request*:



Apabila OTP salah:



Kredit dari user berhasil diubah.

- Proses mendapatkan list tagihan berdasarkan username

http://localhost:8080/api/bill/akuprabu

GET http://localhost:8080/api/bill/akuprabu

Params Authorization Headers (10) Body ● Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL JSON

```

1 {
2   "username": "akuprabu",
3   "amount_credit": 500,
4   "category": "transfer"
5 }

```

Body Cookies Headers (11) Test Results

Status: 200 OK Time: 15 ms Size: 1.22 KB Save Response

Pretty Raw Preview Visualize JSON

```

27   "bill_id": "12",
28   "bill_owner": "akuprabu",
29   "bill_sender": "Tamam",
30   "amount": 1000,
31   "description": "bayar hutang",
32   "isPaid": false,
33   "__v": 0
34 },
35 {
36   "_id": "6098e455536d5844d027cc88",
37   "bill_name": "akuprabu",
38   "bill_id": "1",
39   "bill_owner": "akuprabu",
40   "bill_sender": "test",
41   "amount": 1000,

```

- Proses melakukan *hit* API bila token salah/tanpa token

Saat diberikan token yang salah:

http://localhost:8080/api/bill/

GET http://localhost:8080/api/bill/

Params Authorization Headers (10) Body ● Pre-request Script Tests Settings

Headers 9 hidden

KEY	VALUE	DESCRIPTION	Bulk Edit	Presets
<input checked="" type="checkbox"/> x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6ImVwOTU0YjMxZDUzZTYzMT			
Key	Value	Description		

Body Cookies Headers (11) Test Results

Status: 401 Unauthorized Time: 13 ms Size: 434 B Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   "message": "Unauthorized!"
3 }

```

Saat melakukan akses tanpa token:

http://localhost:8080/api/bill/

GET http://localhost:8080/api/bill/

Params Authorization Headers (10) Body ● Pre-request Script Tests Settings

Headers 9 hidden

KEY	VALUE	DESCRIPTION	Bulk Edit	Presets
<input checked="" type="checkbox"/> x-access-token				
Key	Value	Description		

Body Cookies Headers (11) Test Results

Status: 403 Forbidden Time: 15 ms Size: 436 B Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   "message": "No token provided!"
3 }

```