IF4051 Pengembangan Sistem IoT
Laporan Kerja
UTS

Ng Kyle
13520040

Teknik Informatika
Institut Teknologi Bandung

Laporan Kerja

Daftar Isi

Backend

Setup Backend

1. Backend memanfaatkan Django sebagai Framework Backend dengan bahasa Python
2. Project Backend didesain menjadi bagian:
    a. Definisi model basis data (entitas)
    b. Definisi pub/sub MQTT client
    c. Definisi admin panel
    d. Definisi API untuk User Interface (Frontend)
3. Penggunaan virtual environment untuk membuat lingkungan aplikasi python
4.

```
# membuat virtualenvironment dengan nama venv
python -m venv venv

# menjalankan virtualenvironment
venv/Scripts/activate
```

5. Instalasi package yang digunakan (sudah dicatat dalam requirements.txt)
6.

```
pip install -r requirements.txt
```

7. Membuat project dan app
   Dalam project digunakan:
    - nama_project : paymentapp
    - nama_app : api
    -

```
python manage.py startproject [nama_project]

python manage.py startapp [nama_app]
```

8. Instalasi aplikasi (api) dengan menambahkan INSTALLED_APPS pada paymentapp/settings.py

```
INSTALLED_APPS = [
    ...
    "django.contrib.staticfiles",
    "api.apps.ApiConfig",
]
```

9. Menjalankan server

```
python manage.py runserver –noreload
```

10. Untuk mengakses admin panel pada /admin diperlukan akun superuser yang dapat dibuat dengan command berikut:

```
python manage.py createsuperuser
```

Setup Basis Data

Pada project ini akan digunakan sqlite sebagai engine basis data sehingga tidak perlu mengubah konfigurasi.

Untuk deployment ke cloud, perlu adanya perubahan database engine yang dapat didefinisikan pada DATABASES di patmentapp/settings.py

Model didefinisikan pada file api/models.py

- User : Model untuk mencatat balance dan user yang terdaftar
- TransactionLog : Model untuk mencatat seluruh transaksi yang terjadi

Logic Transaksi Backend

Transaksi dilakukan dengan adanya publish message MQTT “[topic]/deduct’ dengan payload “nim”. MQTT client akan subscribe ke “[topic]/#”

Dalam hal ini topic yang digunakan adalah iot

Semua logic mqtt disimpan pada file paymentapp/mqtt.py yang dikonfigurasi pada paymentapp/__init__.py. File ini akan mendefinisikan sebuah **mqtt client**.

Mqtt client akan memanggil logic transaksi yang didefinisikan pada api/applications/transaction.py.

Secara urut, logic transaksi pada sisi backend diproses sebagai berikut:

1. Message mqtt dibaca oleh mqtt client
2. Mqtt client menjalankan logic transaksi pada aplikasi Transaction
3. Aplikasi Transaction akan melakukan transaksi atomik yaitu:
    - Mengurangi balance dari pengguna

    - Menambahkan log ke pengguna
4. Mqtt client akan melakukan publish ke “iot/success” atau “iot/failed” mengenai hasil dari transaksi

MQTT Broker

Setup Broker

1. MQTT Broker yang digunakan adalah mosquitto
2. Setup konfigurasi pada mosquitto.conf sebagai berikut:

```
listener 1884
listener 8080

allow_anonymous false
password_file passwd.txt

protocol websockets
```

   Konfigurasi tersebut menspesifikasikan bahwa broker akan jalan pada port 1884 dan perlu melakukan autentikasi untuk konek ke broker (allow_anonymous false) dengan list username-password didefinisikan pada passwd.txt

   Listener 8080 dan protocol websockets digunakan agar aplikasi web dapat berkomunikasi dengan protokol ws pada port 8080.

3. Setup password file, membuat file passwd.txt pada folder mosquitto. Lalu pada isinya mencantumkan kumpulan username-password dengan format:

   username:password

   Lalu, file di-format dengan command:

```
mosquitto_passwd -U passwd.txt
```

4. Menjalankan broker

```
mosquitto -c mosquitto.conf
```

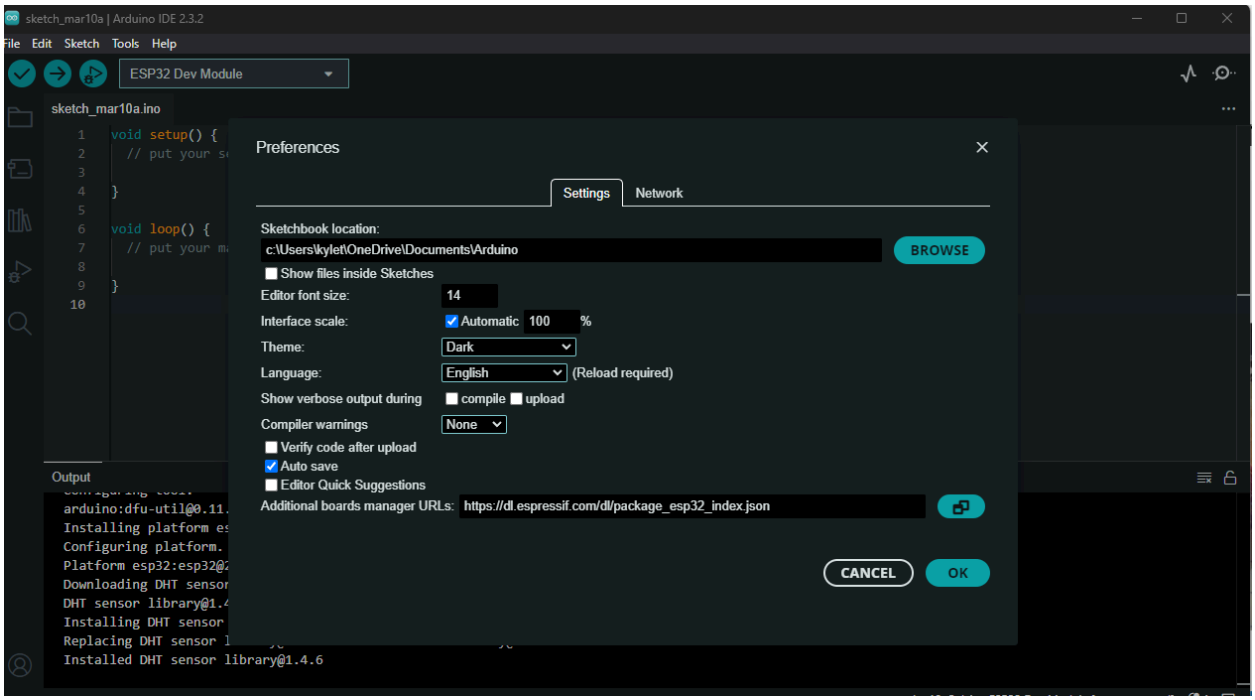
5. Menjalankan subscriber (untuk monitoring)

```
mosquitto_sub -p 1884 -t "iot/#" -u [username] -P [password]
```

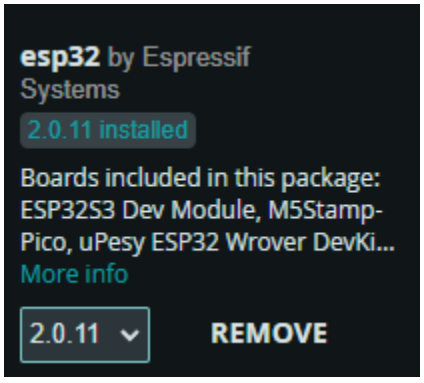
ESP32

Setup Esp32 dan IDE

1. Menggunakan Arduino IDE (Instalasi https://www.arduino.cc/en/software/)
2. Penambahan Additional boards manager pada Arduino IDE untuk ESP32 https://dl.espressif.com/dl/package_esp32_index.json



3. Instalasi board ESP32 pada Boards Manager - esp32 by Espressif



4. Instalasi driver USB to UART untuk komunikasi serial https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers

Logic Transaksi ESP

Kode program:

Sebagai dasar dalam logika pada esp, esp dibagi menjadi 6 blok kode:

1. setup()
   Menyiapkan esp dengan melakukan setup baik PIN, WiFi, dan MQTT Client
2. loop()
   Loop kode utama setelah setup(). Setiap loop dilakukan: Pemeriksaan koneksi mqtt client, update waktu, pemeriksaan input button, dan state LED
3. handleLED()
   Mengatur nyala/mati LED berdasarkan keadaan pembayaran (paymentState).
   PAYMENT_SUCCESS: Lampu menyala selama 5 detik
   PAYMENT_FAILED: Lampu berkedip (dalam interval 200ms) selama 5 detik
4. sendPaymentRequest()
   Mengirimkan request payment dengan publish message ke iot/deduct
5. callback()
   Kode yang dijalankan ketika ada pesan pada MQTT Topic khususnya “iot/success” dan “iot/failed”
6. reconnect()
   Koneksi terhadap MQTT Server

Merchant Reader

Merchant reader merupakan aplikasi yang digunakan untuk menunjukkan status pembayaran saat pembayaran dilakukan.

Setup Merchant Reader

1. Memanfaatkan framework Next dengan Typscript. Menggunakan command:

```
npx create-next-app
```

2. Install mqtt, dengan cara npm install mqtt

```
npm install mqtt
```

3. Kode program layout (UI) utama berada pada page.tsx

4. Kode program untuk mendapatkan data mqtt (MQTT Listener) dan setupnya dideklarasikan pada mqttListener.ts

5. Menjalankan program dengan cara:

```
npm run dev
```

Logic Transaksi Merchant Reader

Aplikasi Merchant Reader subscribe terhadap topic “iot/success” dan “iot/failed”. Setiap data yang didapat diinferensikan berdasarkan topiknya dan menampilkan message bersesuaian selama 5 detik.

Message didefinisikan pada aplikasi Backend. Ketika ada data “iot/success”, merchant akan menampilkan pesan sukses dengan warna hijau. Ketika ada data “iot/failed”, merchant akan menampilkan pesan gagal dengan warna merah.

Pesan juga ditambahkan info saldo untuk menambahkan informasi kepada pengguna.

User App (Dashboard)

User App akan berguna sebagai aplikasi web untuk user melihat daftar transaksi dan juga topup.

Setup User App

1. Memanfaatkan framework Next dengan Typscript. Menggunakan command:

   ```
   npx create-next-app
   ```

2. Install next-auth, dengan cara npm install next-auth

   ```
   npm install next-auth
   ```

3. Logic autentikasi berada pada api/auth/[...nextauth] dengan middlewares.js berfungsi untuk mencatat page yang perlu melakukan login

4. Page yang digunakan didefine pada folder components

5. .env diperlukan dengan isi kurang lebih sebagai berikut

   ```
   NEXTAUTH_URL = "http://localhost:3000"
   NEXTAUTH_SECRET = "secret"
   NEXT_PUBLIC_BACKEND_URL  = "http://localhost:8000"
   ```

Flow User App

1. User melakukan login ke aplikasi pada halaman awal
2. User akan otomatis diredirect ke dashboard jika autentikasi berhasil
3. Dashboard berisikan historis dan balance user.

Authentication

Autentikasi Hardware/Aplikasi - MQTT

Autentikasi antara hardware dan aplikasi dengan mqtt dilakukan pada broker mqtt. Mosquitto sebagai broker akan memeriksa file passwd.txt yang sudah disetup pada Setup Broker. Sehingga koneksi dengan broker mqtt harus dengan signature yang benar (password dan username)

Autentikasi User App

User App melakukan autentikasi dengan pemisahan server-client pada aplikasi next.js. Next-auth diexpose dengan api/auth/[...nextauth]. Bagian client akan merequest autentikasi ke next-auth dan next-auth akan mengembalikan token berupa jwt (JSON web token).

Client akan menyimpan autentikasi pada variable session.

Payment Gateway

Setup Payment Gateway

Payment gateway yang dipakai adalah Midtrans

1. Install helper midtrans

```
npm install midtrans-client
```

2. Install ngrok + setup ngrok
   https://ngrok.com/docs/getting-started/

3. Menyalakan ngrok

```
ngrok http 3000
```

4. Registrasi Midtrans
   https://dashboard.midtrans.com/register

Integrasi Payment Gateway

1. Expose backend pada next.js pada api/tokenizer untuk mendapatkan token transaksi.
2. Pada halaman integrasi UI (UserInfo.js), dilakukan penambahan script dan handler topup

```
useEffect(() => {
        const snapScript = "https://app.sandbox.midtrans.com/snap/snap.js";
        const clientKey = process.env.NEXT_PUBLIC_CLIENT;
        const script = document.createElement("script");

        script.src = snapScript;
        script.setAttribute("data-client-key", clientKey);
        script.async = true;

        document.body.appendChild(script);

        return () => {
            document.body.removeChild(script);
        };
    }, []);
```

```
const handleTopUp = (amount) => {
    const data = {
        id: userInfo?.nim + Date.now(),
        amount: amount,
    };

    fetch(process.env.NEXT_PUBLIC_NGROK_URL + "/api/tokenizer", {
        method: "POST",
        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify(data),
    })
        .then((res) => res.json())
        .then((data_json) => {
            const { token } = data_json;
            window.snap.pay(token, {
                onSuccess: function (result) {
                    // Post to backend
                    fetch(
                        process.env.NEXT_PUBLIC_BACKEND_URL + "/api/topup/",
                        {
                            method: "POST",
                            headers: {
                                "Content-Type": "application/json",
                            },
                            body: JSON.stringify({
                                uid: session?.user?.email,
                                amount: data.amount,
                            }),
                        }
                    );
                },
            });
        });
};
```

Blok kode useEffect akan register script sandboxing midtrans.
Blok kode handleTopUp akan melaksanakan proses topup:

- Membangkitkan snap window untuk payment
- Mengirimkan data topup ke backend jika berhasil

3. Menambahkan beberapa key pada .env

```
NEXT_PUBLIC_CLIENT = SB-Mid-client-xxxxxxxxxx
MIDTRANS_SERVER_KEY = SB-Mid-server-xxxxxxxxxxxx

NEXT_PUBLIC_NGROK_URL = "https://c751-36-90-212-106.ngrok-free.app/"
```

NEXT_PUBLIC_CLIENT digunakan sebagai key publik pada bagian client next.js
MIDTRANS_SERVER_KEY digunakan sebagai key private pada server next.js

Keduanya didapat dari https://dashboard.sandbox.midtrans.com/settings/config_info
Client Key dan Server Key

NEXT_PUBLIC_NGROK_URL merupakan URL yang dibangkitkan ngrok

4. Konek ke user app dengan url ngrok yang sudah dibangkitkan sebelumnya

5. Setup midtrans
   Konfigurasi Finish URL, Unfinish URL, Error Payment URL dengan {NEXT_PUBLIC_NGROK_URL}/thanks
   https://dashboard.sandbox.midtrans.com/settings/snap_preference -> System Settings -> Redirection Settings

6. Untuk melakukan pembayaran, karena sistem development sandboxing, maka dapat digunakan helper berikut: https://simulator.sandbox.midtrans.com/

RFID Scanner

Umumnya, pembayaran sistem IoT mengintegrasi RFID/NFC scanner dalam sistemnya. RFID dibaca dalam bentuk kartu yang menyimpan informasi dari pengguna (akun).

Untuk mengintegrasikan RFID Scanner, pada projek ini digunakan modul RC522.

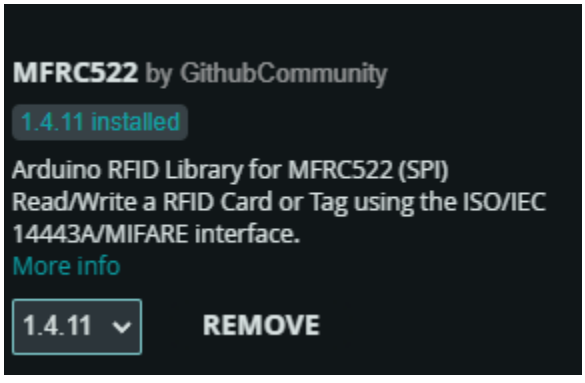
Setup RFID Scanner

1. Scanner RFID memiliki beberapa pin yang perlu dipasang, berikut merupakan rinciannya:

RFID	ESP32
3.3V	3.3V
RST	22
GND	GND
MISO	19
MOSI	23
SCK	18
SDA	5

   IRQ tidak diperlukan dalam projek ini

2. Library RFID, dalam hal ini digunakan



Perubahan kode program ESP

1. Untuk mengintegrasikan RFID, ditambahkan definisi PIN RST dan SS_PIN yang digunakan dalam inisialisasi MRFC522. Kode inisialisasi sebagai berikut (pada void setup)

```
// RFID
SPI.begin();                  // Init SPI bus
mfrc522.PCD_Init();           // Init MFRC522
```

2. Pembacaan RFID dilakukan pada prosedur handleRFID()

```
void handleRFID()
{
  if (mfrc522.PICC_IsNewCardPresent() && mfrc522.PICC_ReadCardSerial())
  {
    String cardUID = "";
    for (byte i = 0; i < mfrc522.uid.size; i++)
    {
      if (mfrc522.uid.uidByte[i] < 0x10)
      {
        cardUID += "0";
      }
      cardUID += String(mfrc522.uid.uidByte[i], HEX);
    }
    cardUID.replace(" ", "");
    cardUID.toUpperCase();
    mfrc522.PICC_HaltA();
    sendPaymentRequest(cardUID);
  }
}
```

Bagian mfrc522.PICC_IsNewCardPresent() && mfrc522.PICC_ReadCardSerial() akan memeriksa adanya kartu yang ditempelkan pada scanner. mfrc522.PICC_HaltA() penting digunakan agar instruksi / blok kode ini hanya akan dijalankan ketika ada kartu yang ditempelkan kembali (menunggu kartu/RFID dilepas dahulu).

Perubahan User Journey

User sebelumnya perlu menekan tombol pada BOOT pada ESP diubah menjadi menempelkan identitas pada RFID scanner.

Perubahan dilakukan pada bagian backend agar dapat menghandle kartu baru berikut juga pada Merchant Reader.