# KEGIATAN LAB MANDIRI

# Implementasi Golang+MVC+React dalam Aplikasi

Pada kegiatan kali ini kita akan mencoba membuat sebuah aplikasi bank sederhana, dengan mengimplementasikan Golang+MVC dari sisi Backend dan React JS dari sisi Frontend.

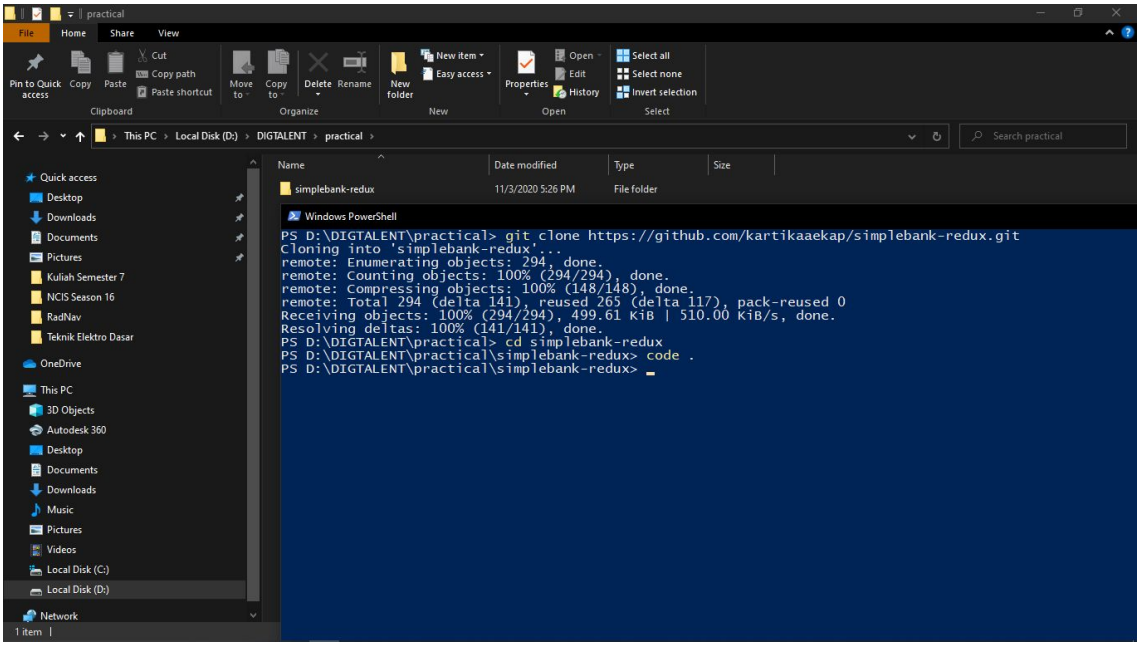
## I. Persiapan Pembuatan Aplikasi Bank Sederhana

### 1. Mengakses Github

a. Akses ke alamat github https://github.com/kartikaaekap/simplebank-redux

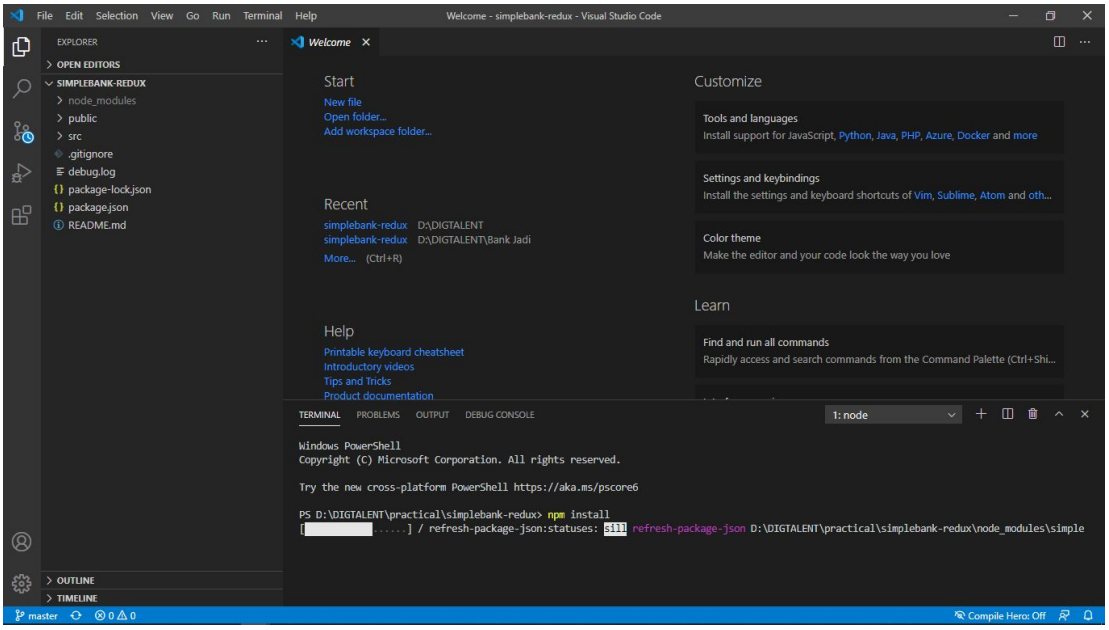
b. Pindah ke halaman File Explorer, pilih lokasi yang diinginkan untuk meng-clone project dari alamat Github di atas. Kemudian, tekan ctrl+shift+klik kanan secara bersamaan, pilih 'open PowerShell (atau Command Prompt) window here'

c. Tulis perintah seperti gambar berikut, untuk melakukan cloning project kemudian membukanya di VSCode.




### 2. Menjalankan Aplikasi Bank Sederhana

a. Buka terminal pada VSCode, kemudian ketik npm install untuk mengunduh node-modules dari apikasi. Setelah selesai, kemudian ketik npm start untuk menjalankan aplikasi




# II. Implementasi API pada React (Membuat Fungsi Transaction)

## 1. Pembuatan File Baru pada Folder Constants dan Actions (Flow Redux)

a. Buat file baru bernama transactionConstants.js di dalam folder constants. Isi file tersebut dengan source code berikut :

```
export const TRANSACTION_DEPOSIT_REQUEST = 'TRANSACTION_DEPOSIT_REQUEST'
export const TRANSACTION_DEPOSIT_SUCCESS = 'TRANSACTION_DEPOSIT_SUCCESS'
export const TRANSACTION_DEPOSIT_FAIL = 'TRANSACTION_DEPOSIT_FAIL'
export const TRANSACTION_DEPOSIT_RESET = 'TRANSACTION_DEPOSIT_RESET'

export const TRANSACTION_WITHDRAWAL_REQUEST = 'TRANSACTION_WITHDRAWAL_REQUEST'
export const TRANSACTION_WITHDRAWAL_SUCCESS = 'TRANSACTION_WITHDRAWAL_SUCCESS'
export const TRANSACTION_WITHDRAWAL_FAIL = 'TRANSACTION_WITHDRAWAL_FAIL'
export const TRANSACTION_WITHDRAWAL_RESET = 'TRANSACTION_WITHDRAWAL_RESET'

export const TRANSACTION_TRANSFER_REQUEST = 'TRANSACTION_TRANSFER_REQUEST'
export const TRANSACTION_TRANSFER_SUCCESS = 'TRANSACTION_TRANSFER_SUCCESS'
export const TRANSACTION_TRANSFER_FAIL = 'TRANSACTION_TRANSFER_FAIL'
export const TRANSACTION_TRANSFER_RESET = 'TRANSACTION_TRANSFER_RESET'

export const TRANSACTION_SALDO_REQUEST = 'TRANSACTION_SALDO_REQUEST'
export const TRANSACTION_SALDO_SUCCESS = 'TRANSACTION_SALDO_SUCCESS'
export const TRANSACTION_SALDO_FAIL = 'TRANSACTION_SALDO_FAIL'
export const TRANSACTION_SALDO_RESET = 'TRANSACTION_SALDO_RESET'
```

b. Kemudian buat file baru bernama transactionActions.js di dalam folder actions. Isi file tersebut dengan source code berikut :

```
import axios from "axios";
import {
    TRANSACTION_DEPOSIT_REQUEST,
    TRANSACTION_DEPOSIT_SUCCESS,
    TRANSACTION_DEPOSIT_FAIL,
    TRANSACTION_WITHDRAWAL_REQUEST,
    TRANSACTION_WITHDRAWAL_SUCCESS,
    TRANSACTION_WITHDRAWAL_FAIL,
    TRANSACTION_TRANSFER_REQUEST,
    TRANSACTION_TRANSFER_SUCCESS,
    TRANSACTION_TRANSFER_FAIL,
    TRANSACTION_SALDO_REQUEST,
    TRANSACTION_SALDO_SUCCESS,
    TRANSACTION_SALDO_FAIL,
  } from "../constants/transactionConstants";
  import { logout } from './userActions'
```

c. Sekarang, kita hendak mengintegrasikan REST API yang telah dibuat, untuk transaksi deposit terlebih dahulu. Tambahkan kode berikut ini pada file transactionActions.js

```
export const deposit = (accountDeposit, amountDeposit, descDeposit) => async (dispatch, getState) => {
  try {
      dispatch({
          type: TRANSACTION_DEPOSIT_REQUEST,
      })

      const {
          userLogin: { token },
      } = getState()

      const config = {
          headers: {
          "Content-Type": "application/json",
          Authorization: `${token}`,
          },
      };
      const { data : {data} } = await axios.post("/api/v1/deposit", {
        transaction_type: 1,
        transaction_description: descDeposit,
        sender: parseInt(accountDeposit),
        recipient: parseInt(accountDeposit),
        timestamp: Date.now(),
        amount: parseInt(amountDeposit)
      }, config)
      dispatch({
          type: TRANSACTION_DEPOSIT_SUCCESS,
          payload: data,
      })
      dispatch(saldo())
```

```
  } catch (error) {
      const message =
        error.response && error.response.data.message
          ? error.response.data.message
          : error.message
      if (message === 'Not authorized, token failed') {
        dispatch(logout())
      }
      dispatch({
        type: TRANSACTION_DEPOSIT_FAIL,
        payload: message,
      })
  }
};
```

d. Selanjutnya, kita akan mengintegrasikan REST API yang telah dibuat, untuk transaksi withdraw. Tambahkan kode berikut ini pada file transactionActions.js dibawah fungsi deposit.

```
export const withdrawal = (accountWithdrawal, amountWithdrawal, descWithdrawal) => async (dispatch, getState) => {
    try {
        dispatch({
            type: TRANSACTION_WITHDRAWAL_REQUEST,
        })

        const {
            userLogin: { token },
        } = getState()

        const config = {
            headers: {
            "Content-Type": "application/json",
            Authorization: `${token}`,
            },
        };
        const { data: {data} } = await axios.post("/api/v1/withdraw", {
        transaction_type: 1,
        transaction_description: descWithdrawal,
        sender: parseInt(accountWithdrawal),
        recipient: parseInt(accountWithdrawal),
        timestamp: Date.now(),
        amount: parseInt(amountWithdrawal)
        }, config);
        dispatch({
            type: TRANSACTION_WITHDRAWAL_SUCCESS,
            payload: data,
        })
        dispatch(saldo())
```

```
    } catch (error) {
      const message =
        error.response && error.response.data.message
          ? error.response.data.message
          : error.message
      if (message === 'Not authorized, token failed') {
        dispatch(logout())
      }
      dispatch({
        type: TRANSACTION_WITHDRAWAL_FAIL,
        payload: message,
      })
    }
};
```

e. Setelah itu, kita akan mengintegrasikan REST API yang telah dibuat, untuk transaksi transfer. Tambahkan kode berikut ini pada file transactionActions.js dibawah fungsi withdraw.

```
export const transfer = (accountTransfer, accountTransferSender, amountTransfer, descTransfer) =>
  async (dispatch, getState) => {
    try {
      dispatch({
        type: TRANSACTION_TRANSFER_REQUEST,
      })

      const {
        userLogin: { token },
      } = getState()

      const config = {
        headers: {
        "Content-Type": "application/json",
        Authorization: `${token}`,
        },
      };
      const { data: {data} } = await axios.post("/api/v1/transfer", {
        transaction_type: 0,
        transaction_description: descTransfer,
        sender: parseInt(accountTransferSender),
        recipient: parseInt(accountTransfer),
        timestamp: Date.now(),
        amount: parseInt(amountTransfer)
       }, config);
      dispatch({
        type: TRANSACTION_TRANSFER_SUCCESS,
        payload: data,
      })
      dispatch(saldo())
```

```
    } catch (error) {
      const message =
        error.response && error.response.data.message
          ? error.response.data.message
          : error.message
      if (message === 'Not authorized, token failed') {
        dispatch(logout())
      }
      dispatch({
        type: TRANSACTION_TRANSFER_FAIL,
        payload: message,
      })
    }
};
```

f. Terakhir, kita akan mengintegrasikan REST API yang telah dibuat, untuk nantinya dapat menampilkan jumlah saldo dari user. Tambahkan kode berikut ini, masih pada file transactionActions.js dibawah fungsi transfer.

```
export const saldo = () => async (dispatch, getState) => {
  try {
    dispatch({
      type: TRANSACTION_SALDO_REQUEST,
    })

    const {
      userLogin: { token },
    } = getState()

    const config = {
      headers: {
        Authorization: `${token}`,
      },
    }

    const { data: {data} } = await axios.get(`/api/v1/account`, config)

    dispatch({
      type: TRANSACTION_SALDO_SUCCESS,
      payload: data,
    })
```

```
  } catch (error) {
    const message =
      error.response && error.response.data.message
        ? error.response.data.message
        : error.message
    if (message === 'Not authorized, token failed') {
      dispatch(logout())
    }
    dispatch({
      type: TRANSACTION_SALDO_FAIL,
      payload: message,
    })
  }
}
```

## 2. Penambahan File Baru pada Folder Reducers

a. Buat file baru bernama transactionReducers.js di dalam folder reducers. Isi file tersebut dengan source code berikut :

```
import {
  USER_LOGIN_FAIL,
  USER_LOGIN_REQUEST,
  USER_LOGIN_SUCCESS,
  USER_LOGOUT,
  USER_REGISTER_FAIL,
  USER_REGISTER_REQUEST,
  USER_REGISTER_SUCCESS,
  USER_REGISTER_STATUS_RESET,
} from "../constants/userConstants";

export const userLoginReducer = (state = {}, action) => {
  switch (action.type) {
    case USER_LOGIN_REQUEST:
      return { loading: true };
    case USER_LOGIN_SUCCESS:
      return { loading: false, token: action.payload };
    case USER_LOGIN_FAIL:
      return { loading: false, error: action.payload };
    case USER_LOGOUT:
      return {};
    default:
      return state;
  }
};
```

```
export const userRegisterReducer = (state = {}, action) => {
  switch (action.type) {
    case USER_REGISTER_REQUEST:
      return { loading: true };
    case USER_REGISTER_SUCCESS:
      return { loading: false, status: action.payload };
    case USER_REGISTER_FAIL:
      return { loading: false, status: action.payload };
    case USER_REGISTER_STATUS_RESET:
      return { loading: false, status: null };
    default:
      return state;
  }
};
```

## 3. Mendaftarkan Reducers yang Telah Dibuat, Pada Store.

Setelah selesai membuat file reducers untuk transaction, selanjutnya kita akan mendaftarkan file tersebut pada store. Tambahkan blok kode berikut pada file store.

```
import { transactionDepositReducer, transactionWithdrawalReducer, transactionTransferReducer, transactionSaldoReducer }
from './reducers/transactionReducers'

const reducer = combineReducers({
  userLogin: userLoginReducer,
  userRegister: userRegisterReducer,
  transactionDeposit: transactionDepositReducer,
  transactionWithdrawal: transactionWithdrawalReducer,
  transactionTransfer: transactionTransferReducer,
  transactionSaldo: transactionSaldoReducer
});
```

4. **Mengintegrasikan Fungsi yang Telah Dibuat, dengan Halaman Transaction**

a. Buka file index.jsx pada folder components > transactions. Kemudian import fungsi transactions pada actions, didalam file tersebut

```
import { deposit, withdrawal, transfer, saldo } from '../../actions/transactionActions';
```

b. Setelah itu, tambahkan kode berikut ini :

```
const [amountDeposit, setAmountDeposit] = useState("");
const [descDeposit, setDescDeposit] = useState("");
const [amountWithdrawal, setAmountWithdrawal] = useState("");
const [descWithdrawal, setDescWithdrawal] = useState("");
const [accountTransfer, setAccountTransfer] = useState("");
const [amountTransfer, setAmountTransfer] = useState("");
const [descTransfer, setDescTransfer] = useState("");

useEffect(() => {
  setAmountDeposit("");
  setDescDeposit("");
  setAmountWithdrawal("");
  setDescWithdrawal("");
  setAccountTransfer("");
  setAmountTransfer("");
  setDescTransfer("");
}, [])

useEffect(() => {
  if (token) {
    dispatch(saldo())
  }
}, [dispatch, history, token])

const transactionSaldo = useSelector((state) => state.transactionSaldo)
const { saldoTotal } = transactionSaldo
const accountDeposit = saldoTotal?.account?.account_number
const accountWithdrawal = saldoTotal?.account?.account_number
const accountTransferSender = saldoTotal?.account?.account_number
```

```
const submitDepositHandler = (e) => {
  e.preventDefault();
  dispatch(deposit(accountDeposit, amountDeposit, descDeposit));
  dispatch(saldo)
};
const submitWithdrawalHandler = (e) => {
  e.preventDefault();
  dispatch(withdrawal(accountWithdrawal, amountWithdrawal, descWithdrawal));
  dispatch(saldo)
};
const submitTransferHandler = (e) => {
  e.preventDefault();
  dispatch(transfer(accountTransfer, accountTransferSender, amountTransfer, descTransfer));
  dispatch(saldo)
};
```

c. Setelah itu, kita akan menambahkan jumlah saldo user pada halaman transactions. Tambahkan blok kode berikut pada masing-masing tab untuk deposit, withdraw, dan transfer

```
<div className="mb-5">
  <h4>Total Saldo : {saldoTotal != null && saldoTotal.account ? saldoTotal.account.saldo : 0} </h4>
</div>
```

d. Terakhir, tambahkan fungsi onSubmit={} pada masing-masing form transactions. Jangan lupa juga untuk menambahkan value seperti contoh berikut :

```
<Form.Group as={Row} controlId="formPlaintextAmount">
  <Form.Label column sm="2">
    Total Amount
  </Form.Label>
  <Col sm="10">
    <Form.Control
      type="amount"
      value={amountDeposit}
      onChange={(e) => setAmountDeposit(e.target.value)}
      placeholder="Input the amount" />
  </Col>
</Form.Group>
```