Combining Front-End and Back-End

Fullstack Vs. Front-End (FE) + Back-End (BE)

- Untuk development web saat ini terdapat 2 style, yaitu adanya pemisahan antara front-end (FE) dan back-end (BE) (membuat 2 project / repository terpisah) atau fullstack (FE dan BE langsung jadi 1 project / repository)
- Untuk membuat website dengan ReactJs dengan style fullstack, bisa digunakan framework lain yang diintegrasikan dengan react. Contoh yang populer adalah Next.js dan Remix.js

Fullstack Vs. Front-End (FE) + Back-End (BE)

- Untuk membuat website dengan ReactJs dengan style BE + FE, kita membuat website dengan React seperti biasa untuk FE dan membuat sebuah web service untuk BE
- Untuk kuliah ini kita akan menggunakan React sebagai FE dan Express sebagai BE.

Starter Code (BE)

- dibagikan di shared praktikan
- menggunakan express dan sequelize
- buat DB `fpwinf20231_m5`
- install dependency dan jalankan migration dan seeder

```
npm install
npx sequelize-cli db:migrate
npx sequelize-cli db:seed:all
```

Starter Code (FE)

- install dependency

```
npm install react-hook-form joi @hookform/resolvers axios
```

```jsx
// App.jsx
import { useState } from "react";
import ContactForm from "./ContactForm";
import ContactItem from "./ContactItem";

const host = "http://localhost:3000/api";

export default function App() {
  const [contacts, setContacts] = useState([
    { id: "onhmq", name: "DUMMY", phone: "08100000000", favorite: true },
    { id: "37occ", name: "DUMMY", phone: "08111111111", favorite: false },
    { id: "thi5i", name: "DUMMY", phone: "08122222222", favorite: true },
  ]);
  const [activeContact, setActiveContact] = useState({});
  const [filter, setFilter] = useState("");

  const formMethods = {
    submit: function (data) {
      console.log(data);
    },
    reset: function () {
      setActiveContact({ id: "", name: "", phone: "", favorite: false });
    },
  };

  const contactMethods = {
    edit: function (id) {
      const c = contacts.find((x) => x.id == id);
      console.log(c);
      if (c) {
        setActiveContact(c);
      }
    },
  };

  function filterChange(e) {
    setFilter(e.target.value);
  }

  return (
    <div>
      <div>
        <input
          type="search"
          name="q"
          onChange={filterChange}
          placeholder="search"
          value={filter}
        ></input>
      </div>
      <br />
      <div>
        {contacts.length > 0 ? (
          contacts.map((c) => (
            <ContactItem
              contactMethods={contactMethods}
              key={c.id}
              contact={c}
            />
          ))
        ) : (
          <>
            <h3>Tidak ada kontak</h3>
            <hr />
          </>
        )}
      </div>
      <div>
        <ContactForm activeContact={activeContact} formMethods={formMethods} />
      </div>
    </div>
  );
}
```

```jsx
//ContactItem.jsx
export default function ContactItem({ contact, contactMethods }) {
  return (
    <>
      <div>
        Name:{contact.name}
        <br />
        Phone:{contact.phone}
        <br />
        Action:
        <button>{contact.favorite ? "💖" : "🖤"}</button>
        <button onClick={() => contactMethods.edit(contact.id)}>📝</button>
        <button>🗑️</button>
      </div>
      <hr />
    </>
  );
}
```

```jsx
// ContactForm.jsx
import { useForm } from "react-hook-form";
import Joi from "joi";
import { joiResolver } from "@hookform/resolvers/joi";

export default function ContactForm({ activeContact, formMethods }) {
  const schema = Joi.object({
    id: Joi.string().allow(""),
    name: Joi.string().required(),
    phone: Joi.string().required(),
    favorite: Joi.boolean().default(false),
  });
  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm({
    values: activeContact,
    resolver: joiResolver(schema),
  });

  return (
    <div>
      <form onSubmit={handleSubmit(formMethods.submit)}>
        <input type="hidden" {...register("id")} />
        <table>
          <tbody>
            <tr>
              <td>Name</td>
              <td>:</td>
              <td>
                <input {...register("name")} />
                {errors.name ? (
                  <>
                    <br />
                    <span>{errors.name.message}</span>
                  </>
                ) : (
                  ""
                )}
              </td>
            </tr>
            <tr>
              <td>Phone</td>
              <td>:</td>
              <td>
                <input {...register("phone")} />
                {errors.phone ? (
                  <>
                    <br />
                    <span>{errors.phone.message}</span>
                  </>
                ) : (
                  ""
                )}
              </td>
            </tr>
            <tr>
              <td colSpan={2}>
                <input type="submit" value="Insert" />
                <button
                  onClick={(e) => {
                    e.preventDefault();
                    formMethods.reset();
                  }}
                >
                  Clear selection
                </button>
              </td>
            </tr>
          </tbody>
        </table>
      </form>
    </div>
  );
}
```

useEffect

- Hook yang menerima sebuah function
- Function dijalankan setelah semua component selesai di-render. Sekilas mirip dengan event listener onLoad tetapi berbeda dengan onLoad.
- Fungsinya adalah untuk sinkronisasi halaman web yang diatur oleh React dengan sistem eksternal seperti web service atau plugin yang tidak menggunakan react (misalnya library jquery)

useEffect

- Pakailah hanya jika benar-benar diperlukan
  - You might not need an effect: contoh kasus di mana kita tidak perlu effect
  - Intinya: hanya pakai useEffect jika kita benar-benar perlu untuk synchronization dengan komponen eksternal (di luar React)
  - Banyak kasus di mana kita bisa menggantikan useEffect dengan event atau teknik lain. Masing-masing kasus nanti akan dijelaskan waktu praktik jika ditemukan.

Syntax useEffect

```
useEffect(setup, dependencies?)
```

- `setup` : Function yang hendak dijalankan. Return nya boleh undefined atau sebuah function lain.
  - Apabila return function lain, maka function tersebut menjadi `cleanup function` (akan dijelaskan nanti)
  - **Tidak boleh async function**.

Syntax useEffect

```
useEffect(setup, dependencies?)
```

- `dependencies` : array yang bisa diisi dengan prop, state, dan variable atau function yang dideklarasikan di komponen. Function setup akan dijalankan ulang apabila terjadi perubahan value pada salah satu variabel yang ada dalam array.

Syntax useEffect

`useEffect(setup, dependencies?)`

```
useEffect(() => {
  const connection = createConnection(serverUrl, roomId);
  connection.connect();
  return () => {
    connection.disconnect();
  };
}, [serverUrl, roomId]);
```

Dependency useEffect

```
useEffect(() => {
  // This runs after every render
});

useEffect(() => {
  // This runs only on mount (when the component appears)
}, []);

useEffect(() => {
  // This runs on mount _and also_ if either a or b have changed since the last render
}, [a, b]);
```

Studi Kasus CRUD Contacts

Membaca dan menampilkan kontak dari web service

GET data dengan useEffect + Axios

Jangan begini ya

```
useEffect(async () => {
  const temp = await axios.get("http://localhost:3000/api/contacts");
  console.log(temp.data);
  setContacts(temp.data.contacts);
});
```

GET data dengan useEffect + Axios

```
useEffect(() => {
  axios.get("http://localhost:3000/api/contacts").then(function (x) {
    console.log(x.data);
    setContacts(x.data.contacts);
  });
});
```

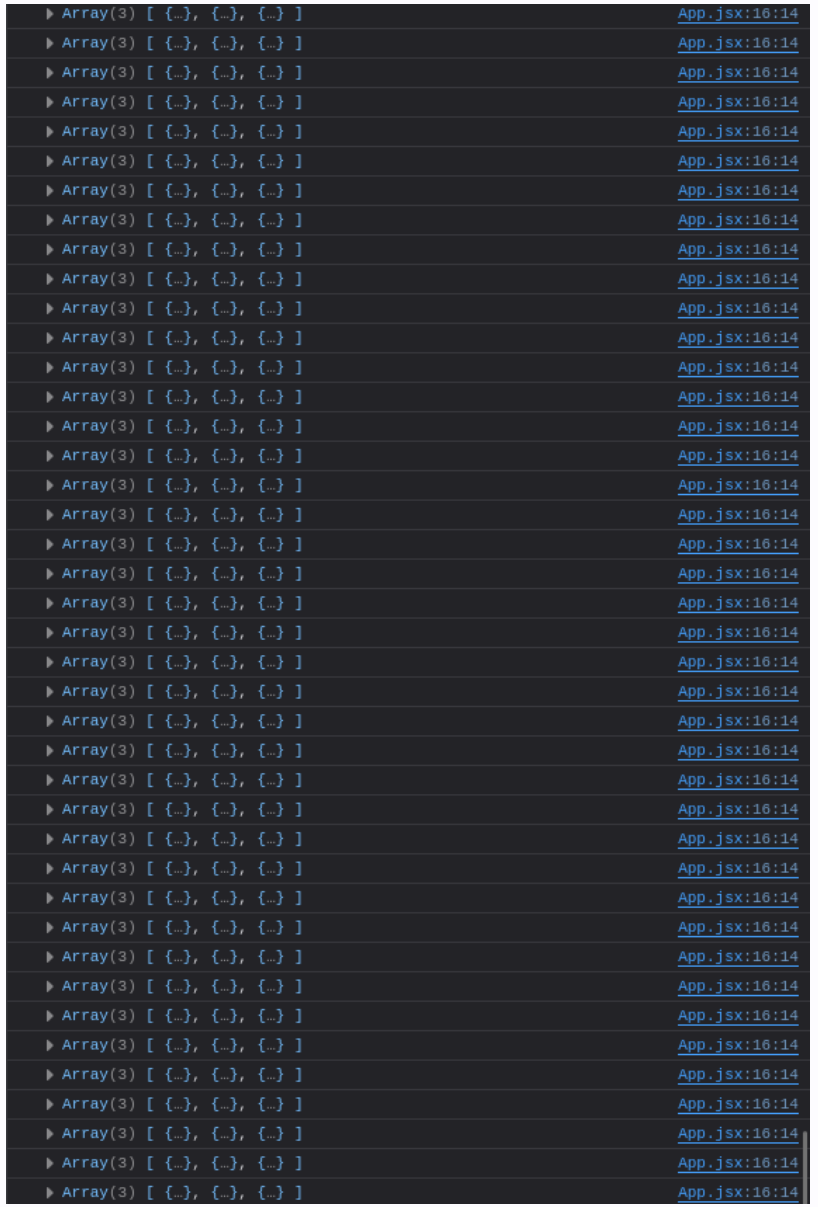
CORS

Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at *http://localhost:3000/api/contacts*. (Reason: CORS header 'Access-Control-Allow-Origin' missing). Status code: 200. [Learn More]

Mengatasi CORS (BE)

```
//index.js
app.use(
  cors({
    origin: "http://localhost:5173",
    optionsSuccessStatus: 200,
  })
);
```

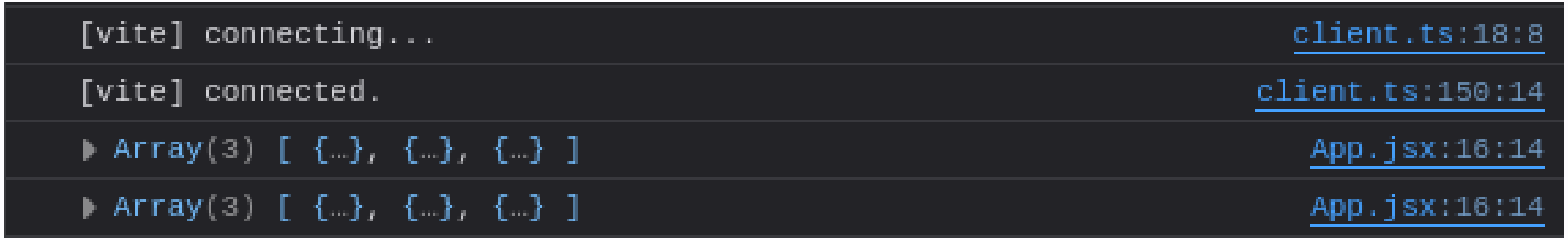
Request non-stop



Buat supaya useEffect jalan 1x saja

```
useEffect(() => {
  axios.get("http://localhost:3000/api/contacts").then(function (x) {
    console.log(x.data);
    setContacts(x.data);
  });
}, []);
```

useEffect jalan 2x



useEffect jalan 2x

- Abaikan, memang benar begini. Di mode development React menjalankan useEffect 2x.
- Ini untuk mengetes bahwa useEffect sudah dijalankan dengan benar.
- Kalau menjalankan useEffect 2x merusak app kalian:
  - cleanup function lupa diimplementasikan, atau
  - cleanup function salah implementasi, atau
  - function ini tidak cocok untuk jadi effect, bisa diimplementasikan sebagai event atau diakali dengan metode lain

useEffect jalan 2x

- Cleanup function dijalankan ketika useEffect hendak dijalankan ulang. Jadi kalau cleanup function diberikan maka akan dijalankan effect, cleanup, lalu effect lagi.
- Kalau cleanup function diimplementasikan dengan benar, maka effect dijalankan sekali hasilnya akan sama dengan menjalankan effect + cleanup + effect

Tambah contact baru

```
const formMethods = {
  submit: function (data) {
    if (!data.id) {
      axios.post(`${host}/contacts`, data).then(function (response) {
        setContacts([...contacts, response.data]);
      });
    }
  },
  //...
};
```

Tambah contact jangan pakai useEffect

- Tambah contact harusnya terjadi ketika user menekan tombol submit, sehingga seharusnya tambah contact adalah event
- Ini contoh kesalahan yang ditemukan oleh React dengan menjalankan useEffect 2x: kalau contact dijadikan effect, maka data akan masuk ke DB 2x.

Hapus contact

Delete contact (BE)

```
// ./src/controllers/contact.js
//...
module.exports = {
  //...
  delete: async function (req, res) {
    const contact = await db.Contact.findByPk(req.params.id);
    if (contact) {
      await contact.destroy();
      return res.status(200).send(contact);
    } else {
      return res.status(404).send({ msg: "not found" });
    }
  },
};
```

Delete contact (FE)

```
//App.jsx
const contactMethods = {
  //...
  delete: function (id) {
    axios.delete(`${host}/contacts/${id}`).then(function (response) {
      setContacts(contacts.filter((c) => c.id !== id));
    });
  },
};
//ContactItem.jsx
export default function ContactItem({ contact, contactMethods }) {
  return (
    //...
    <button onClick={() => contactMethods.delete(contact.id)}>🗑</button>
    //...
  );
}
```

Optimistic rendering

- Update state di FE dengan menganggap bahwa operasi di BE sukses. Jika terjadi kegagalan maka BE akan memberi tahu FE dan FE akan mengembalikan state ke keadaan sebelumnya
- Kekurangan: data di FE bisa tidak sync dengan data di BE
- Kelebihan: UX jauh lebih baik, terutama apabila jaringan jelek

Delete contact (FE) + Optimistic rendering

```
const contactMethods = {
  //...
  delete: function (id) {
    setContacts(contacts.filter((c) => c.id !== id));
    axios.delete(`${host}/contacts/${id}`).catch(function (err) {
      console.log(err);
      alert("error, check console");
      // ini mengembalikan ke posisi awal
      setContacts(contacts);
      // ingat: setContact sebelum axios.delete tidak mengubah
      // isi state contacts yang akan dipakai di callback ini
    });
  },
};
```

Mencari contact berdasarkan nama atau telepon

Search contact

```
useEffect(() => {
  axios.get(`${host}/contacts?q=${filter}`).then(function (x) {
    console.log(x.data);
    setContacts(x.data.contacts);
  });
}, [filter]);
```

Kenapa ini effect, bukan event?

- Search query biasanya ditaruh di url. Kalau misal user menekan tombol back, ada kemungkinan search query di url berubah tapi event onChange tidak dijalankan
- Di studi kasus ini sebenarnya url di browser tidak berubah. Code menggunakan effect diajarkan hanya untuk membiasakan diri.

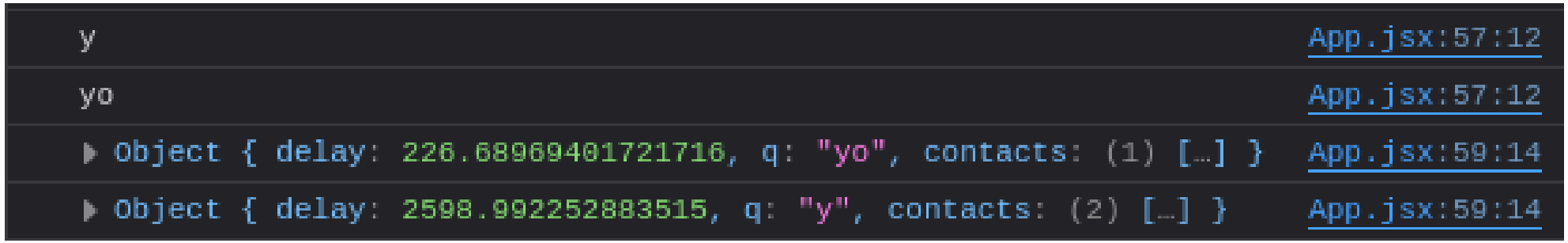
Race condition

- Endpoint search di BE telah dimodifikasi untuk mensimulasikan network yang tidak stabil. Response didelay secara random.

```
module.exports = {
  get: async function (req, res) {
    try {
      //...
      const delay = Math.random() * 3000;
      setTimeout(() => res.status(200).send({ delay, q, contacts }), delay);
    } catch (err) {
      return res.status(500).send(err);
    }
  },
  //...
};
```

Race condition

- Karena adanya delay ini, jika kita mengetik "yo", ada kemungkinan response dari search "yo" datang duluan sebelum response dari search "y"



Race condition

- Akibatnya bisa jadi search result yang ditampilkan adalah untuk "y", bukan "yo"

- React seharusnya mengabaikan hasil search "y" karena kita punya nilai search yang baru ("yo")

Cleanup function

- kita bisa menambahkan cleanup function yang meminta React untuk mengabaikan hasil pengambilan data apabila sudah data yang baru

```
useEffect(() => {
  let ignore = false;
  axios.get(`${host}/contacts?q=${filter}`).then(function (x) {
    if (!ignore) {
      setContacts(x.data.contacts);
    }
  });
  return () => {
    ignore = true;
  };
}, [filter]);
```

Silakan coba sendiri membuat fitur-fitur berikut

- update contact
- toggle favorite