

Dokumentasi Projek Pemrograman Web Berorientasi Layanan

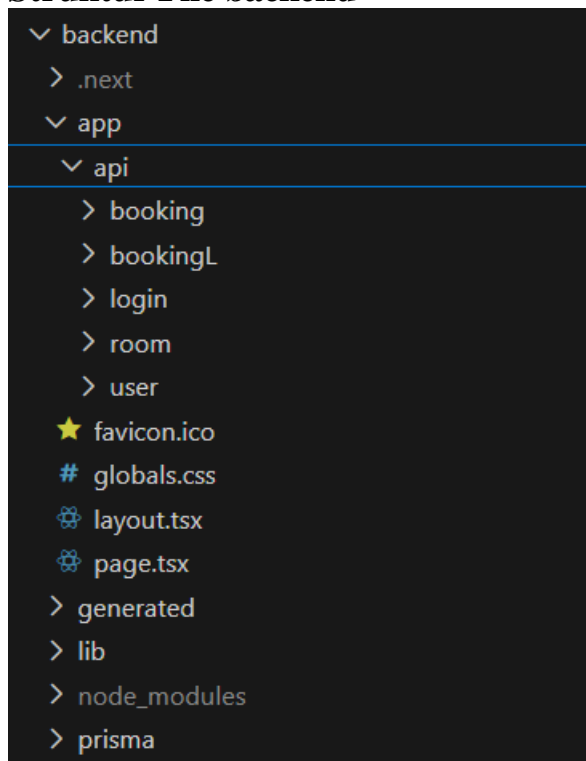
# **IMPLEMENTASI ROOMSYNC SEBAGAI SISTEM RESERVASI HOTEL BERBASIS ARSITEKTUR API**



Disusun Oleh:  
Nama : Valbian Alfikri  
NPM : 23313019

**PROGRAM STUDI S1 TEKNOLOGI INFORMASI  
FAKULTAS TEKNIK DAN ILMU KOMPUTER  
UNIVERSITAS TEKNOKRAT INDONESIA  
2026**

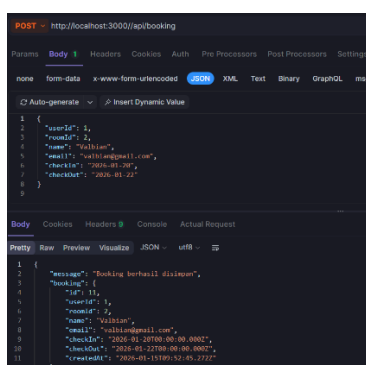
## 1. Struktur File backend



Struktur folder ini dibuat mengikuti standar **Next.js App Router** agar aplikasi tersusun rapi dan mudah dipahami. Folder `app` berfungsi sebagai pusat pengaturan routing, sedangkan `app/api` digunakan khusus untuk pengelolaan backend API. Setiap folder seperti `user`, `login`, `room`, dan `booking` merepresentasikan satu fitur atau endpoint tertentu, sehingga kode tidak tercampur, lebih terstruktur, dan mudah dikelola dalam proses pengembangan aplikasi.

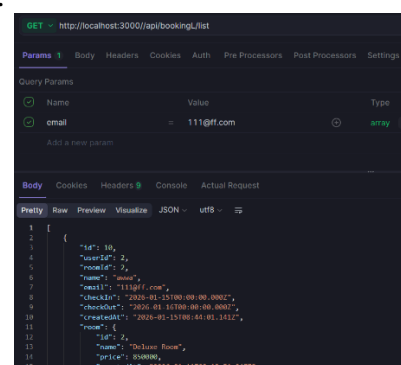
## 2. API Booking

Hasil :



Melakukan pemesanan menggunakan nama email yang sudah pernah daftar sebelumnya, bisa melakukan pemesanan kamar dengan room id 2 – 3 lalu menyertakan tanggal check-in dan check-out. Menampilkan pesan error dan memblokir button jika tanggal check-in sesudah tanggal check-out.

Hasil :



Menampilkan Riwayat booking yang sudah dipesan dari akun yang sedang digunakan untuk memesan kamar.

Code :

```
// check-out harus lebih besar dari check-in
if (new Date(checkOut) <= new Date(checkIn)) {
  return NextResponse.json({
    error: "Tanggal check-out harus setelah check-in",
  }, {
    status: 400, headers: corsHeaders
  });
}

// Menyimpan data booking ke database menggunakan Prisma
const booking = await prisma.booking.create({
  data: {
    // userId dikonversi ke Number karena biasanya dikirim sebagai string dari form
    userId: Number(userId), // Foreign key ke tabel user

    // roomId juga dikonversi ke Number
    roomId: Number(roomId), // Foreign key ke tabel room

    // Nama pemesan kamar
    name: name,

    // Email bersifat opsional, jika tidak ada maka disimpan null
    email: email ?? null,

    // Konversi string tanggal ke tipe Date
    checkIn: new Date(checkIn),
    checkOut: new Date(checkOut),
  },
});
```

Code :

```
export async function GET(req: NextRequest) {
  try {
    // Mengambil query parameter dari URL
    // Contoh URL: /api/bookings/list?email=111@ff.com
    const { searchParams } = new URL(req.url);

    // Mengambil nilai email dari query parameter
    const email = searchParams.get("email");

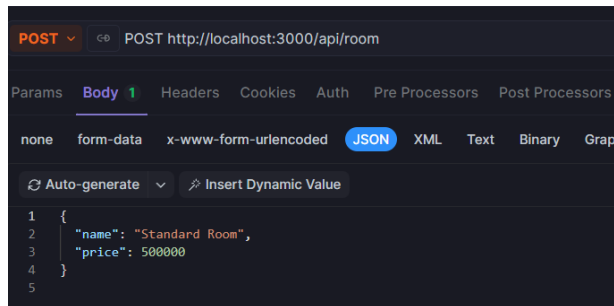
    // Validasi: email wajib dikirim
    // Jika tidak ada email, maka request dianggap tidak valid
    if (!email) {
      return NextResponse.json({
        error: "Email wajib dikirim",
        status: 400, headers: corsHeaders // status 400 = bad request
      });
    }

    // Mengambil data booking dari database menggunakan Prisma
    // Data difilter berdasarkan email
    const bookings = await prisma.booking.findMany({
      where: {
        email: email, // hanya booking dengan email ini yang diambil
      },
      include: {
        room: true, // ikut ambil data relasi room
        user: true, // ikut ambil data relasi user
      },
      orderBy: {
        createdAt: "desc", // urutkan dari data terbaru ke terlama
      },
    });

    // Mengirim response sukses berupa data booking
    // Disertai header CORS agar bisa diterima frontend
    return NextResponse.json(bookings, { headers: corsHeaders });
  }
}
```

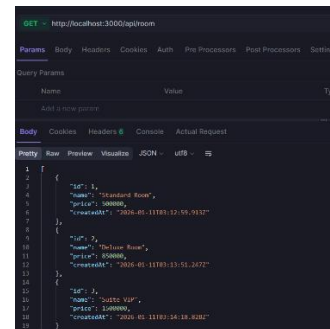
### 3. API Room

Hasil :



Menambahkan kamar dan harga sesuai dengan keinginan untuk mendapatkan id kamar, lalu Id kamar dapat diakses untuk melakukan booking.

Hasil :



Data kamar yang sudah dibuat akan digunakan untuk melakukan booking menggunakan Id. Id yang digunakan adalah 2 – 3 yang dimana merupakan *Deluxe Room* dan *Suits VIP*.

Code :

```
// Fungsi ini digunakan untuk menambahkan data kamar baru ke database
export async function POST(req: Request) {
  try {
    // Mengambil data JSON dari body request yang dikirim frontend
    const { name, price } = await req.json();

    // Validasi: nama kamar dan harga wajib diisi
    // Jika salah satu kosong, kirim response error
    if (!name || !price) {
      return NextResponse.json({
        error: "Nama kamar dan harga wajib diisi",
        status: 400 // 400 = bad request
      });
    }

    // Menyimpan data kamar baru ke database menggunakan Prisma
    const room = await prisma.room.create({
      data: {
        name, // nama kamar
        price: Number(price) // harga kamar dikonversi ke Number
      },
    });
  }
}
```

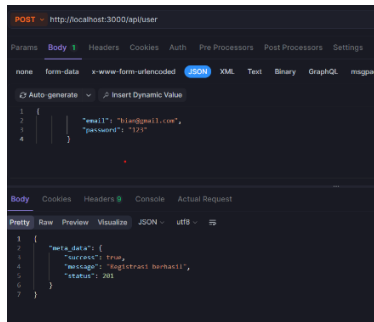
Code :

```
// Fungsi ini digunakan untuk mengambil semua data kamar dari database
export async function GET() {
  // Mengambil seluruh data kamar dari tabel room
  const rooms = await prisma.room.findMany();

  // Mengirim data kamar ke frontend dalam bentuk JSON
  return NextResponse.json(rooms);
}
```

## 4. API Login dan Register

Hasil :



Menambahkan fungsi post untuk melakukan registrasi pada halaman website yang akan dibuat dan menampilkan pesan saat email yang digunakan sudah pernah log-in sebelumnya. Memberikan enkripsi untuk password yang diinput saat mendaftar.

Code :

```
export async function POST(request: NextRequest) {
  try {
    // Mengambil data JSON dari body request
    const body: UserRequest = await request.json();

    // Validasi: email dan password wajib diisi
    if (!body.email || !body.password) {
      return NextResponse.json(
        {
          meta_data: {
            success: false,
            message: "Email dan password wajib diisi",
            status: 400,
          },
        },
        { status: 400, headers: corsHeaders }
      );
    }

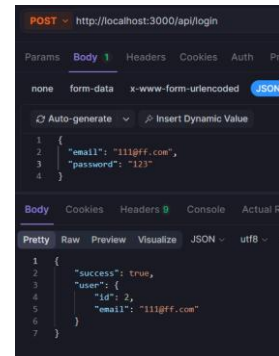
    // Mengecek apakah email sudah terdaftar di database
    const check = await prisma.tb_user.findFirst({
      where: { email: body.email },
    });

    // Jika email sudah ada, registrasi ditolak
    if (check) {
      return NextResponse.json(
        {
          meta_data: {
            success: false,
            message: "Email sudah digunakan",
            status: 409,
          },
        },
        { status: 409, headers: corsHeaders } // 409 = conflict
      );
    }

    // Meng-hash password sebelum disimpan ke database
    // Angka 10 adalah salt rounds untuk keamanan
    const hashedPassword = await bcrypt.hash(body.password, 10);

    // Menyimpan user baru ke database
    await prisma.tb_user.create({
      data: {
        email: body.email,
        password: hashedPassword, // simpan password dalam bentuk hash
      },
    });
  }
}
```

Hasil :



Mengirim email dan password pada database dan melakukan kecocokan email dan password untuk dapat mengakses halaman yang akan di tuju. Mengirim pesan error jika email tidak terdaftar pada database website. Mengirim pesan error jika password tidak pas dengan email yang terdaftar.

Code :

```
export async function POST(request: NextRequest) {
  try {
    // Mengambil email dan password dari body JSON yang dikirim frontend
    const { email, password } = await request.json();

    // Validasi: email dan password wajib diisi
    // Jika salah satu kosong, langsung kirim error ke frontend
    if (!email || !password) {
      return NextResponse.json(
        {
          message: "Email dan password wajib diisi",
          status: 400, headers: corsHeaders } // 400 = bad request
      );
    }

    // Mencari user di database berdasarkan email
    const user = await prisma.tb_user.findFirst({
      where: { email },
    });

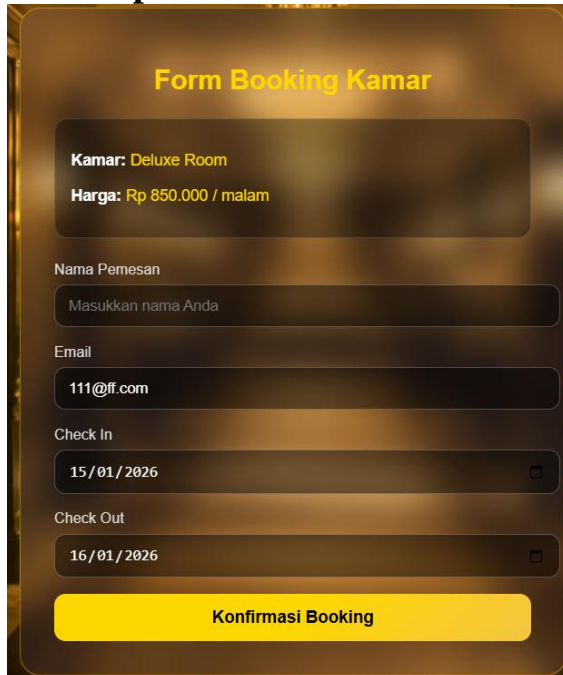
    // Jika user tidak ditemukan, berarti email belum terdaftar
    if (!user) {
      return NextResponse.json(
        {
          message: "Email tidak terdaftar",
          status: 404, headers: corsHeaders } // 404 = data tidak ditemukan
      );
    }

    // Membandingkan password yang diinput dengan password hash di database
    // bcrypt.compare akan mengembalikan true jika cocok, false jika tidak
    const match = await bcrypt.compare(password, user.password);

    // Jika password tidak cocok
    if (!match) {
      return NextResponse.json(
        {
          message: "Password salah",
          status: 401, headers: corsHeaders } // 401 = unauthorized
      );
    }

    // Jika email dan password benar, kirim response sukses
    return NextResponse.json(
      {
        success: true, // penanda bahwa login berhasil
        user: {
          id: user.id, // kirim ID user
          email: user.email // kirim email user
          // Data sensitif seperti password tidak dikirim ke frontend
        }
      }
    );
  }
}
```

## 5. Form Update



The image shows a mobile app interface for a room booking form. At the top, it says 'Form Booking Kamar'. Below this, there are two rows of information: 'Kamar: Deluxe Room' and 'Harga: Rp 850.000 / malam'. Underneath, there are four input fields: 'Nama Pemesan' with a placeholder 'Masukkan nama Anda', 'Email' with '111@ff.com', 'Check In' with '15/01/2026', and 'Check Out' with '16/01/2026'. At the bottom, there is a large yellow button labeled 'Konfirmasi Booking'.

Menambahkan Nama Pemesan dan Email pada form booking agar dapat disimpan ke dalam database supaya dikenali dan bisa dihubungi. Menyambungkan button Konfirmasi Booking dengan route di folder booking.

Code :

```
/* Input Nama Pemesan */
<div style={fieldStyle}>
  <label style={labelStyle}>Nama Pemesan</label>
  <input
    type="text"
    style={inputStyle}
    value={name}
    placeholder="Masukkan nama Anda"
    onChange={(e) => setName(e.target.value)}
  />
</div>

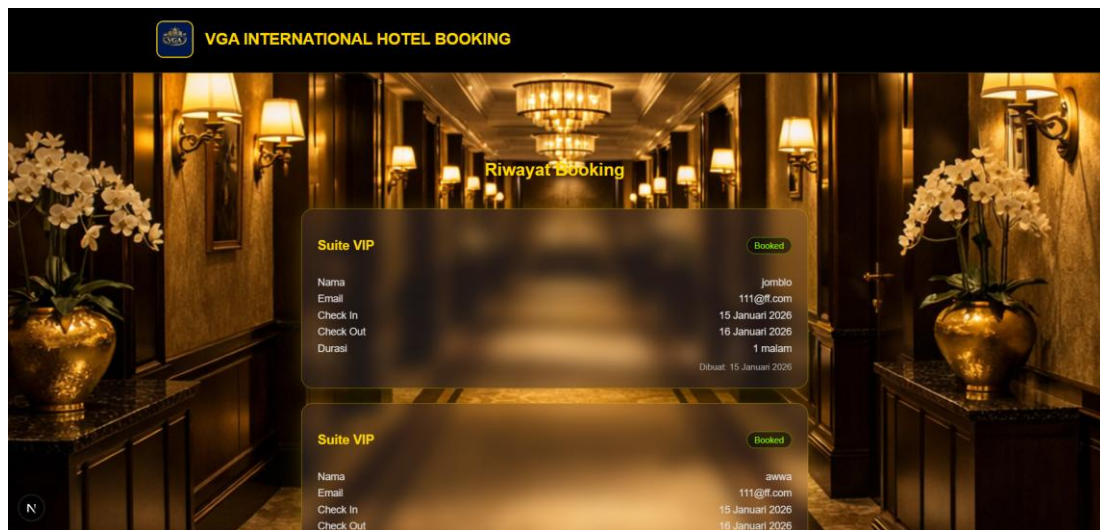
/* Input Email (opsional) */
<div style={fieldStyle}>
  <label style={labelStyle}>Email</label>
  <input
    type="email"
    style={inputStyle}
    value={email}
    placeholder="Masukkan email (opsional)"
    onChange={(e) => setEmail(e.target.value)}
  />
</div>
```

```
// Kirim data booking ke API backend
const res = await fetch("http://localhost:3000/api/booking", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  // Body dikirim dalam format JSON
  body: JSON.stringify({
    userId: user.id, // dari auth
    roomId: room.id, // dari mapping ROOM_DATA
    name: name, // dari input form (wajib)
    email: email, // dari input / user
    checkIn: checkIn,
    checkOut: checkOut,
  }),
});
```

Link POST agar tersambung dengan database

Menambahkan pengisian nama dan email, email harus sesuai dengan yang terdaftar.

## 6. Update Riwayat Booking



Menyambungkan halaman Riwayat dengan file API list agar halaman dapat menampilkan data Riwayat booking dari akun email yang melakukan pemesanan.

Code :

```
// Panggil API bookingL/list dengan parameter email
// Jadi backend hanya mengembalikan booking milik email ini saja
fetch(`http://localhost:3000/api/bookingL/list?email=${user.email}`)
  .then((r) => r.json()) // Ubah response menjadi JSON
  .then((res) => {
    console.log("Data booking dari API:", res); // Debug di console
    setData(res); // Simpan data booking ke state
    setLoading(false); // Matikan status loading
  })
  .catch((err) => {
    console.error("Gagal mengambil data booking:", err);
    setLoading(false);
  });
}, [user?.email]);
// useEffect akan dijalankan ulang jika email user berubah
```