Nama : Alexander Radianta Tarigan

No\_Peserta : FSDO003ONL010

Link Heroku : <https://alex-payment-api.herokuapp.com/>

Link github : https://github.com/alextarigan/alex-payment-api.git

Create database [XYBkUrHis2](https://remotemysql.com/phpmyadmin/db_structure.php?server=1&db=XYBkUrHis2) di remotemysql.com

Pembuatan database onlineshop di mysql lokal

1. Pembuatan class AuthResult.cs pada folder Configuration

using System.Collections.Generic;

namespace PaymentAPI.Configuration

{

    public class AuthResult

    {

        public string Token {get; set;}

        //public string RefreshToken {get; set;}

        public bool Success {get; set;}

        public List<string> Errors {get; set;}

    }

}

2. Pembuatan class JwtConfig.cs pada folder Configuration

using System.Collections.Generic;

namespace PaymentAPI.Configuration

{

    public class JwtConfig

    {

        public string Secret {get; set;}

    }

}

3. Pembuatan class AuthManagementControllers pada folder Controllers

a. Pembuatan methode register (untuk melakukan register akun, dan akan menggenerate sebuah token)

[HttpPost]

        [Route("Register")]

        public async Task<IActionResult> Register([FromBody] UserRegistrationDto user)

        {

            if(ModelState.IsValid)

            {

                var existingUser = await \_userManager.FindByEmailAsync(user.Email);

                if(existingUser != null)

                {

                    return BadRequest(new RegistrationResponse(){

                        Errors = new List<string>(){

                            "Email Telah digunakan"

                        },

                        Success = false

                    });

                }

                var newUser = new IdentityUser() { Email = user.Email, UserName = user.Username };

                var isCreated = await  \_userManager.CreateAsync(newUser, user.Password);

                if(isCreated.Succeeded)

                {

                    var jwtToken = GenerateJwtToken( newUser);

                    return Ok(new RegistrationResponse() {

                        Success = true,

                        Token = jwtToken

                    });

                }else{

                    return BadRequest(new RegistrationResponse(){

                        Errors = isCreated.Errors.Select(x => x.Description).ToList(),

                        Success = false

                    });

                }

            }

            return BadRequest(new RegistrationResponse(){

                Errors = new List<string>()  {

                    "Invalid Payload"

                },

                Success = false

            });

        }

b. Pembuatan methode login(akan menghasil token yang nantinya akan digunakan sebagai bearer token untuk autentikasi)

[HttpPost]

        [Route("Login")]

        public async Task<IActionResult> Login([FromBody] UserLoginRequest user)

        {

            if(ModelState.IsValid)

            {

                var existingUser = await \_userManager.FindByEmailAsync(user.Email);

                if(existingUser == null){

                    return BadRequest(new RegistrationResponse(){

                        Errors = new List<string>{

                            "Invalid login request"

                        },

                        Success = false

                    });

                }

                var isCorrect = await \_userManager.CheckPasswordAsync(existingUser, user.Password);

                if(!isCorrect){

                    return BadRequest(new RegistrationResponse(){

                        Errors = new List<string>(){

                            "Invalid login request"

                        },

                        Success= false

                    });

                }

                var jwtToken = GenerateJwtToken(existingUser);

                return Ok(new RegistrationResponse(){

                    Success = true,

                    Token = jwtToken

                });

            }

            return BadRequest(new RegistrationResponse(){

                Errors = new List<string>(){

                    "Invalid payload"

                },

                Success = false

            });

        }

c. Pembuatan methode GenerateJwtToken ( yang akan digunakan untuk autentikasi)

private string GenerateJwtToken(IdentityUser user)

        {

            var jwtTokenHandler = new JwtSecurityTokenHandler();

            var key= Encoding.ASCII.GetBytes(\_jwtConfig.Secret);

            var tokenDescriptor = new SecurityTokenDescriptor

            {

                Subject = new ClaimsIdentity(new []

                {

                    new Claim("Id", user.Id),

                    new Claim(JwtRegisteredClaimNames.Email, user.Email),

                    new Claim(JwtRegisteredClaimNames.Sub, user.Email),

                    new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())

                }),

                Expires = DateTime.UtcNow.AddHours(6),

                SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)

            };

            var token = jwtTokenHandler.CreateToken(tokenDescriptor);

            var jwtToken = jwtTokenHandler.WriteToken(token);

            return jwtToken;

        }

4. Pembuatan Controller Payment pada folder Controllers dan menambahkan Authentication agar saat ingin menggunakan controller diperlukan jwt token yang didapat Ketika berhasil login, dan berlaku selama 30 menit.

[Route("api/[controller]")]

    [ApiController]

    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

a. Membuat methode Get(untuk mendapatkan semua data)

[HttpGet]

        public async Task<IActionResult> GetItem()

        {

            var payments = await \_context.Payments.ToListAsync();

            return Ok(payments);

        }

b. Membuat methode GetById (untuk mendapatkan data berdasarkan paymentDetailId)

[HttpGet("{id}")]

        public async Task<IActionResult> GetItem(int id)

        {

            var payment = await \_context.Payments.FirstOrDefaultAsync(x => x.paymentDetailId == id);

            if(payment == null)

                return NotFound();

            return Ok(payment);

        }

c. Membuat methode Post Data (menambahkan data ke dalam database)

[HttpPost]

        public async Task<IActionResult> CreateItem(Payment data)

        {

            if(ModelState.IsValid)

            {

                await \_context.Payments.AddAsync(data);

                await \_context.SaveChangesAsync();

                return CreatedAtAction("GetItem", new {data.paymentDetailId}, data);

            }

            return new JsonResult("Something went wrong") {StatusCode = 500};

        }

d. Membuat methode put data (untuk mengubah data yang ada didalam database)

[HttpPut("{id}")]

        public async Task<IActionResult> UpdateItem(int id, Payment payment)

        {

            if(id != payment.paymentDetailId)

                return BadRequest();

            var existItem = await \_context.Payments.FirstOrDefaultAsync(x => x.paymentDetailId == id);

            if(existItem == null)

                return NotFound();

            existItem.cardOwnerName = payment.cardOwnerName;

            existItem.cardNumber = payment.cardNumber;

            existItem.expirationDate = payment.expirationDate;

            existItem.securityCode = payment.securityCode;

            await \_context.SaveChangesAsync();

            return NoContent();

        }

e. Membuat methode Delete data (untuk menghapus data yang ada didalam database berdasarkan paymentDetailId)

[HttpDelete("{id}")]

        public async Task<IActionResult> DeleteItem(int id)

        {

            var existItem = await \_context.Payments.FirstOrDefaultAsync(x => x.paymentDetailId == id);

            if(existItem == null)

                return NotFound();

            \_context.Payments.Remove(existItem);

            await \_context.SaveChangesAsync();

            return Ok(existItem);

        }

5. Pembuatan class ApiDbContext.cs pada folder Data

public class ApiDbContext : IdentityDbContext

    {

        public virtual DbSet<Payment> Payments {get; set;}

        public ApiDbContext(DbContextOptions<ApiDbContext> options)

            :base(options)

        {

        }

    }

6. Membuat class Payment.cs pada folder Models

public class Payment

    {

        [Key]

        public int paymentDetailId {get; set;}

        public string cardOwnerName {get; set;}

        public string cardNumber {get; set;}

        public DateTime expirationDate {get; set;}

        public string securityCode{get; set;}

    }

7. melakukan setting koneksi database di appsetings.json

{

  "ConnectionStrings": {

    "DefaultConnection": "Server=remotemysql.com;Port=3306;Database=XYBkUrHis2;Uid=XYBkUrHis2;Pwd=IG4CuwLT7p;SSL Mode=none"

  },

  "Logging": {

    "LogLevel": {

      "Default": "Information",

      "Microsoft": "Warning",

      "Microsoft.Hosting.Lifetime": "Information"

    }

  },

  "JwtConfig": {

    "Secret": "guczlcbrpiluwsydtdnzdixpmwbdjibr"

  },

  "AllowedHosts": "\*"

}

8. Membuat service untuk koneksi database di class Startup.cs

services.AddDbContext<ApiDbContext>(options =>

                options.UseMySql(

                    Configuration.GetConnectionString("DefaultConnection"),new MySqlServerVersion(new Version())

                ));

9. Membuat authorize untuk menginput bearer token

c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

                {

                    Description = @"JWT Authorization header using the Bearer scheme. \r\n\r\n

                      Enter 'Bearer' [space] and then your token in the text input below.

                      \r\n\r\nExample: 'Bearer 12345abcdef'",

                    Name = "Authorization",

                    In = ParameterLocation.Header,

                    Type = SecuritySchemeType.ApiKey,

                    Scheme = "Bearer"

                });

                c.AddSecurityRequirement(new OpenApiSecurityRequirement()

                {

                    {

                        new OpenApiSecurityScheme {

                            Reference = new OpenApiReference {

                                Type = ReferenceType.SecurityScheme,

                                Id = "Bearer"

                            }

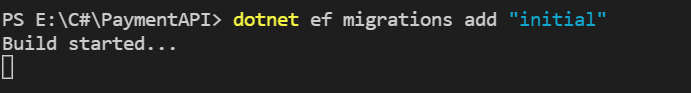
                        },

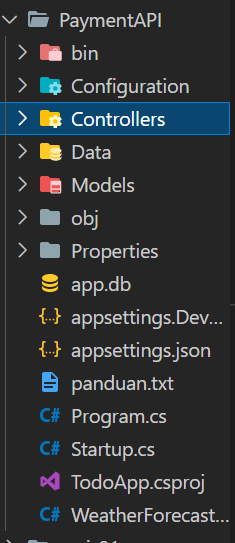
                        new string[] { }

                    }

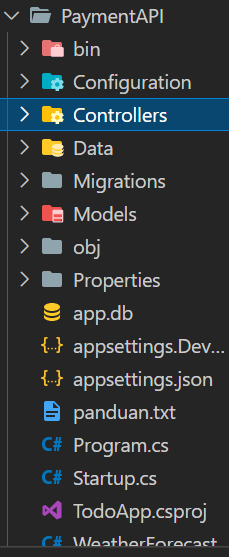
                });

Melakukan migrasi database

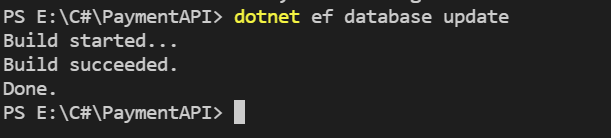




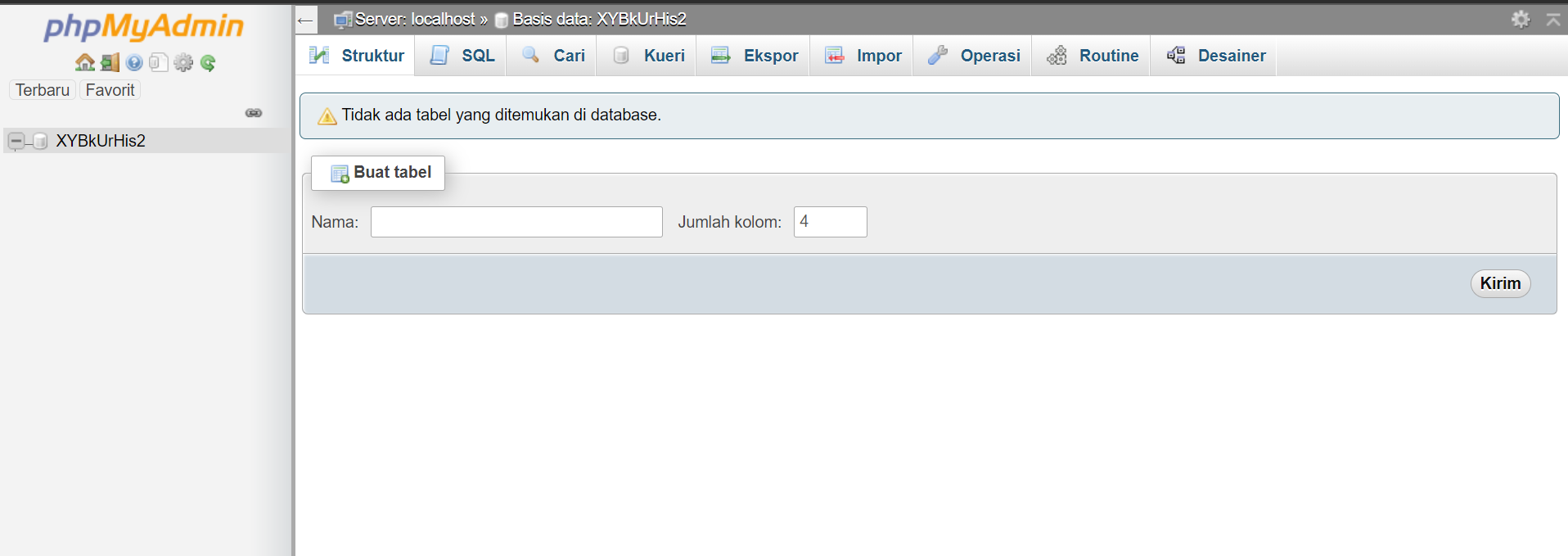
Setelah dilakukan migrasi, maka akan membuat sebuah folder migrations



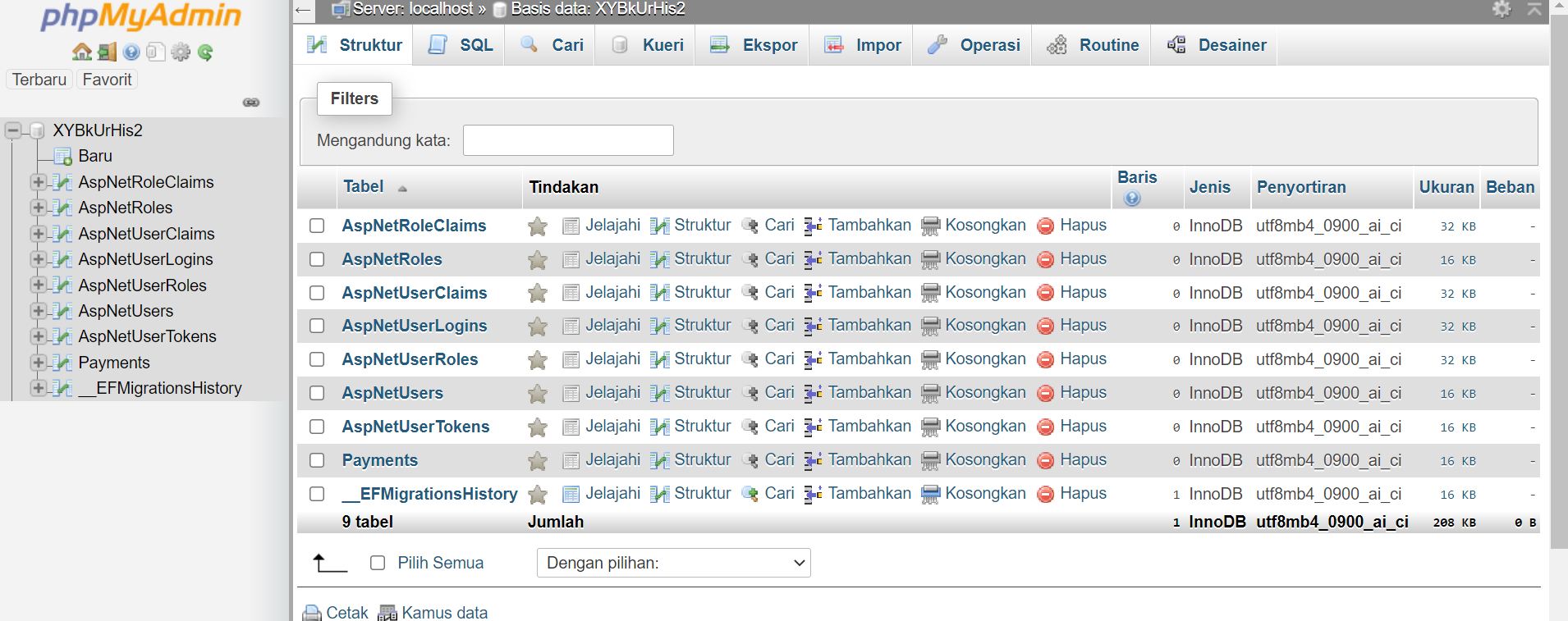
Melakukan database update



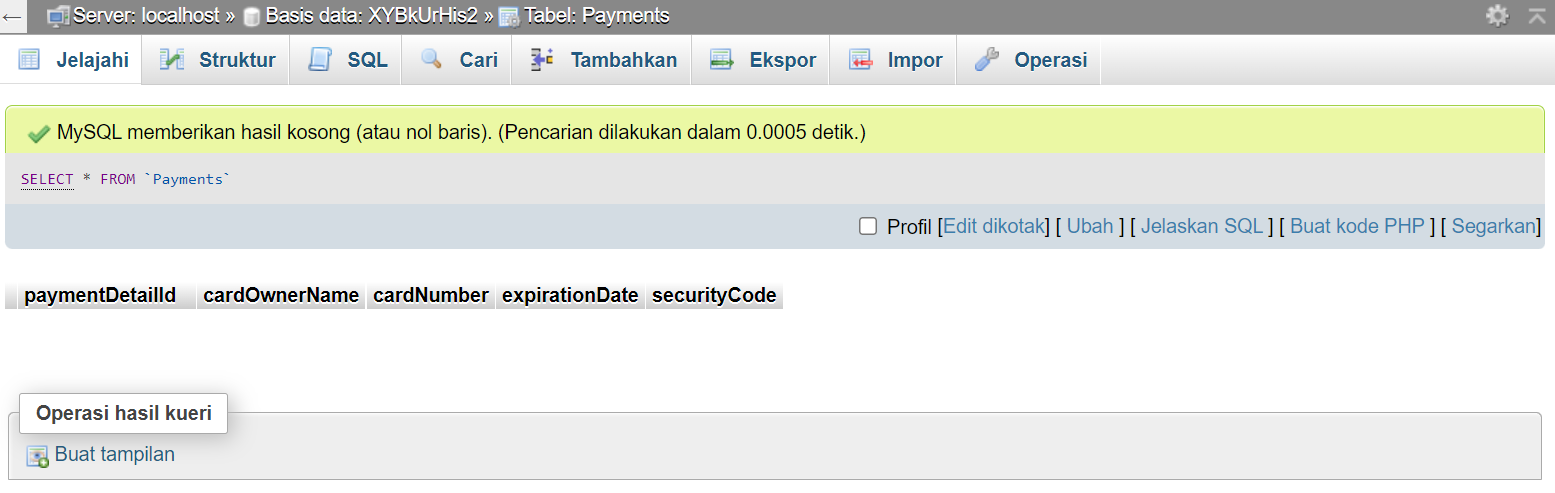
Sebelum dilakukan database update (database masih kosong)



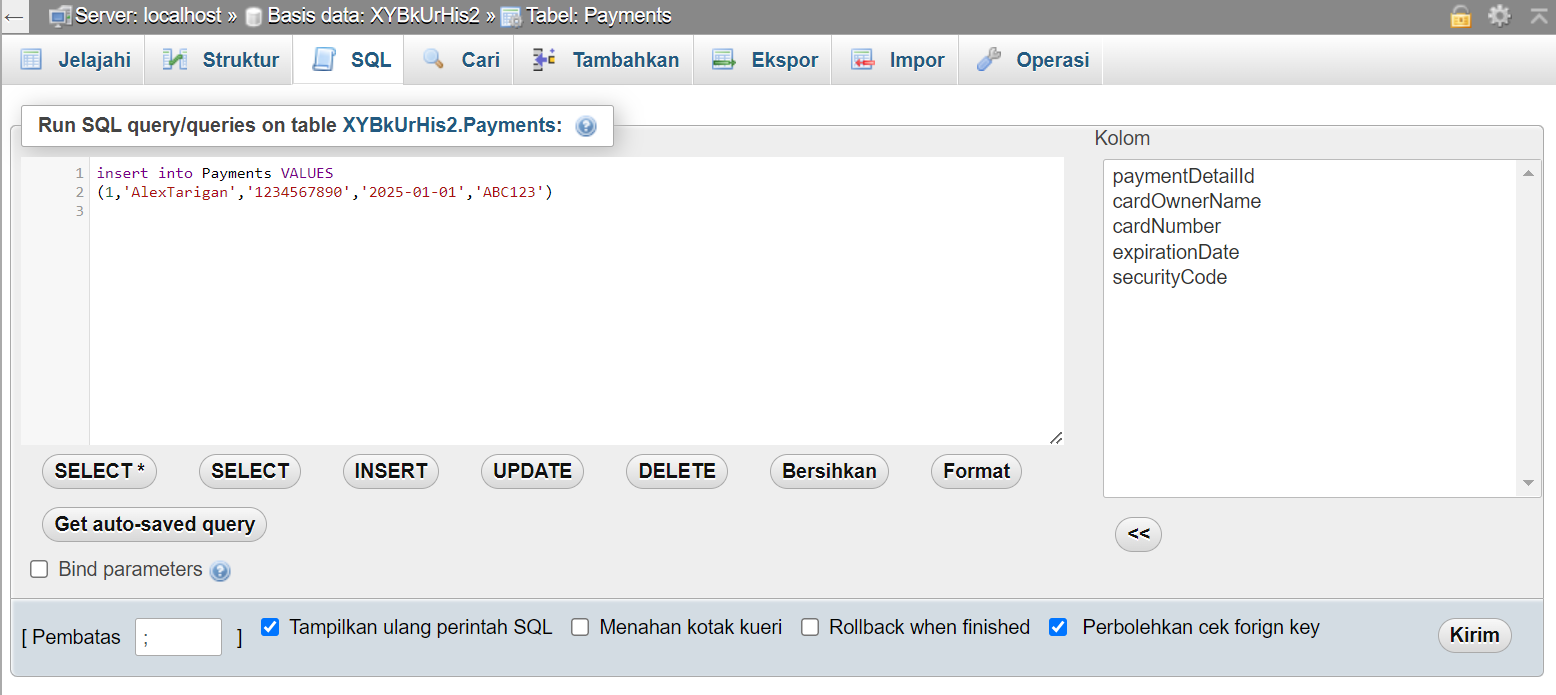
Setelah dilakukan database update (database memiliki beberapa table)



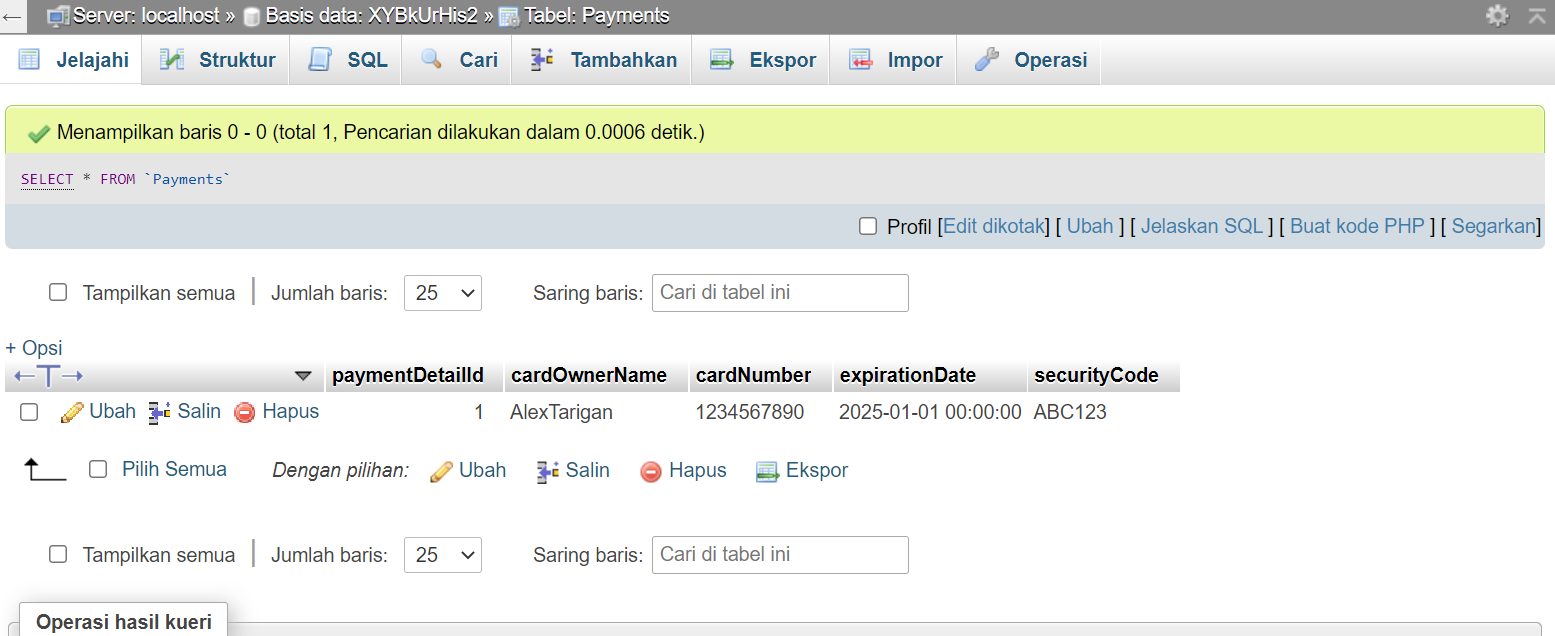
Tabel Payments masih kosong



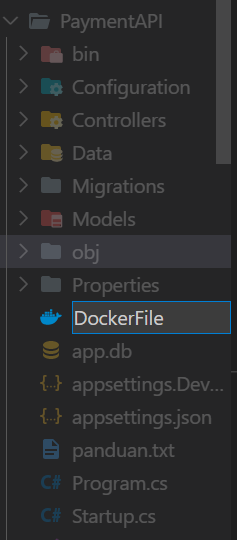
Lakukan Insert 1 data ke dalam table Payments



1 data berhasil ditambahkan ke table payments



Buat file Dockerfile



#See https://aka.ms/containerfastmode to understand how Visual Studio uses this Dockerfile to build your images for faster debugging.

FROM mcr.microsoft.com/dotnet/aspnet:5.0 AS base

WORKDIR /app

EXPOSE 80

EXPOSE 443

EXPOSE 48719

EXPOSE 27017

EXPOSE 5000

EXPOSE 5001

FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build

WORKDIR /src

COPY . .

#heroku menggunakan cli

#COPY ["TodoAppTest/TodoAppTest.csproj", "TodoAppTest/"]

#RUN dotnet restore "TodoAppTest/TodoAppTest.csproj"

#COPY . .

#WORKDIR "/src/TodoAppTest"

#RUN dotnet build "TodoAppTest.csproj" -c Release -o /app/build

RUN dotnet restore

RUN dotnet build --no-restore -c Release -o /app

FROM build AS publish

RUN dotnet publish "PaymentAPI.csproj" -c Release -o /app/publish

FROM base AS final

WORKDIR /app

#COPY --from=publish /app/publish .

COPY --from=publish /app/ .

#ENTRYPOINT ["dotnet", "TodoAppTest.dll"]

CMD ASPNETCORE\_URLS=http://\*:$PORT dotnet PaymentAPI.dll

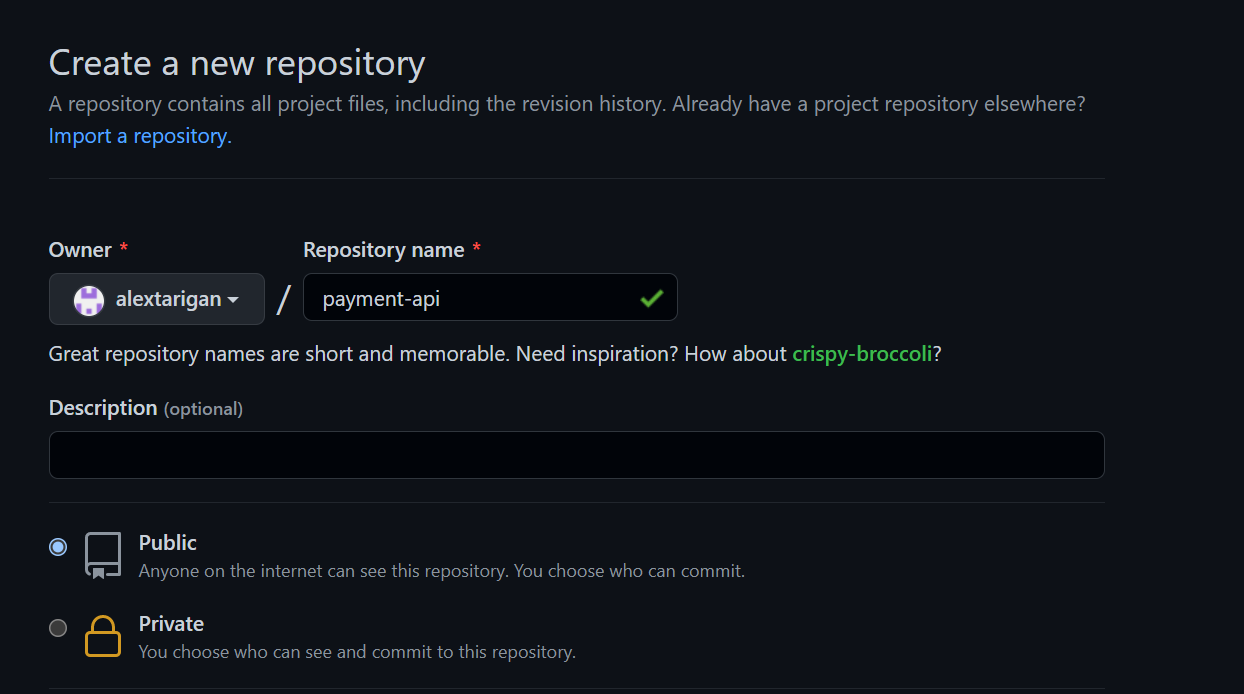
Lalu buka terminal dan jalankan dotnet build



Lalu jalankan dotnet publish -c Release



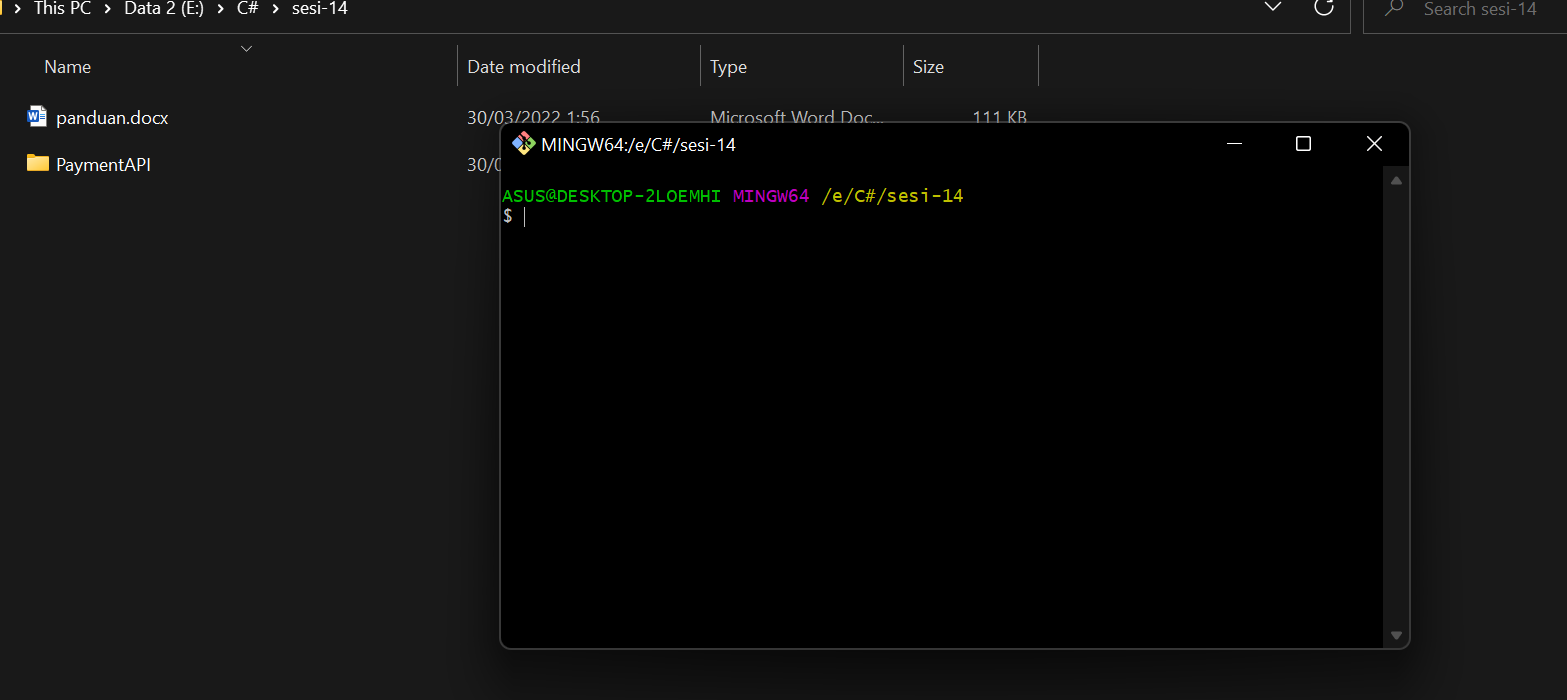
Buat repository baru di github



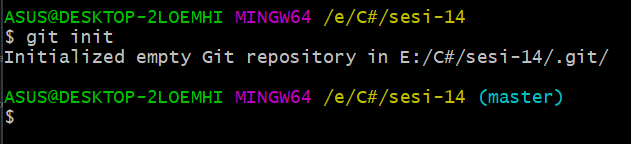
Repository baru berhasil dibuat



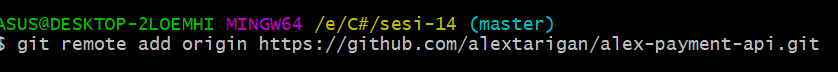
Buka git bash di folder final project



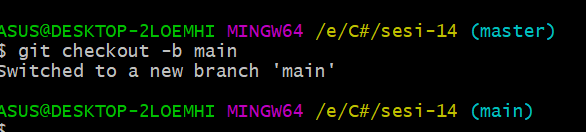
Ketikkan perintah git init



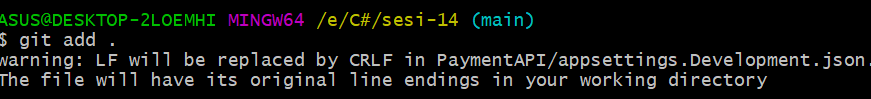
Ketikkan perintah git remote add origin “link repository”



Ketikka perintah git checkout -b main



Ketikkan perintah git add .



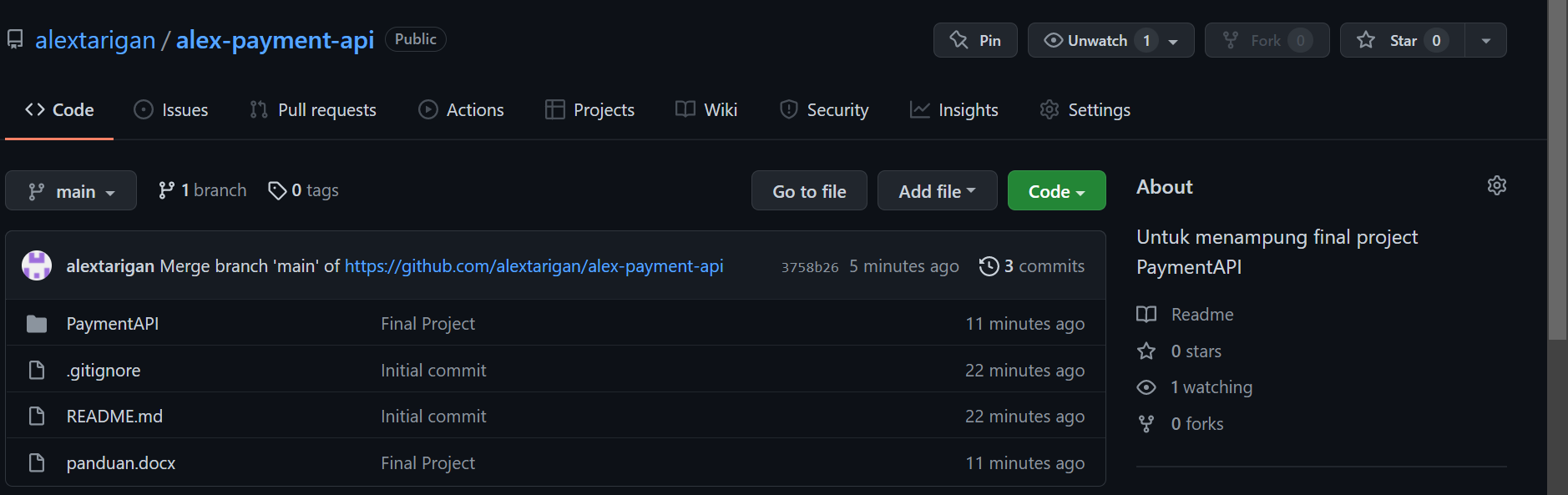
Ketikkan perintah commit -m “final project”



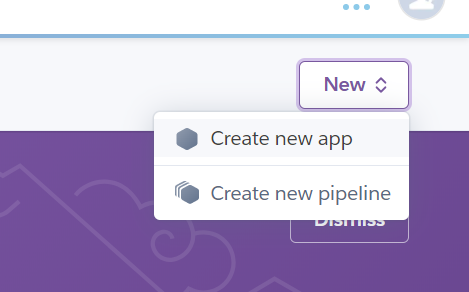
Ketikkan perintah git push –set-upstream origin main



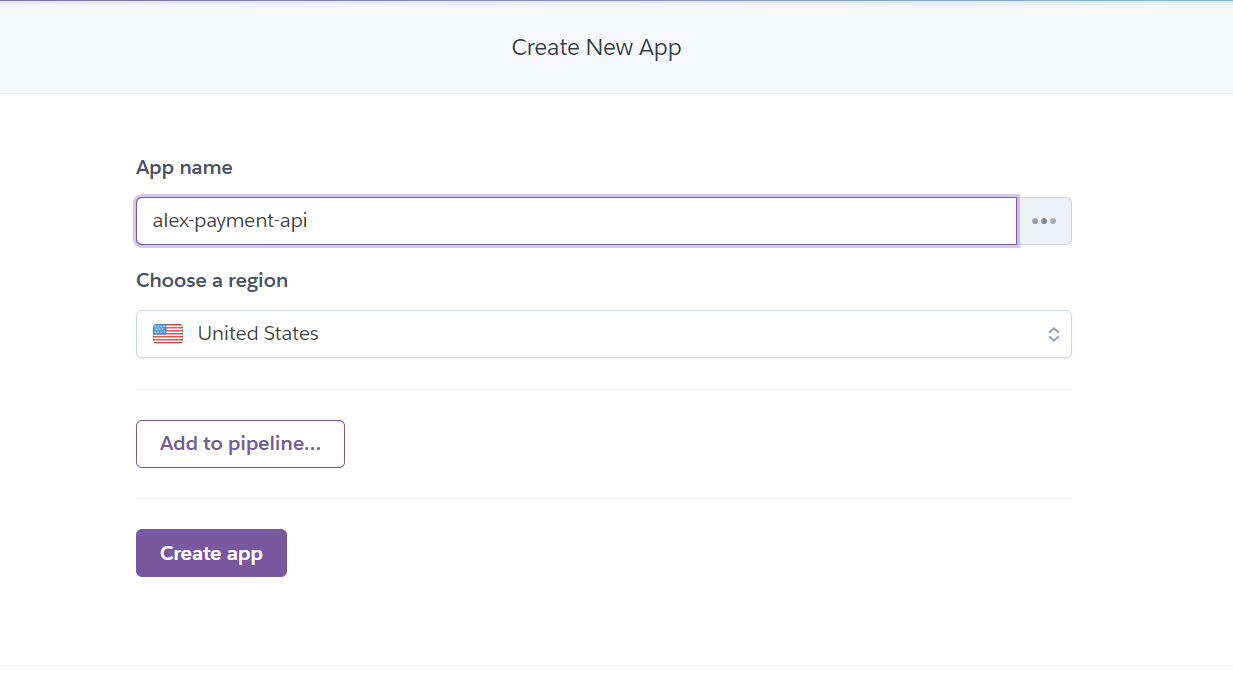
File berhasil di push ke repo di github



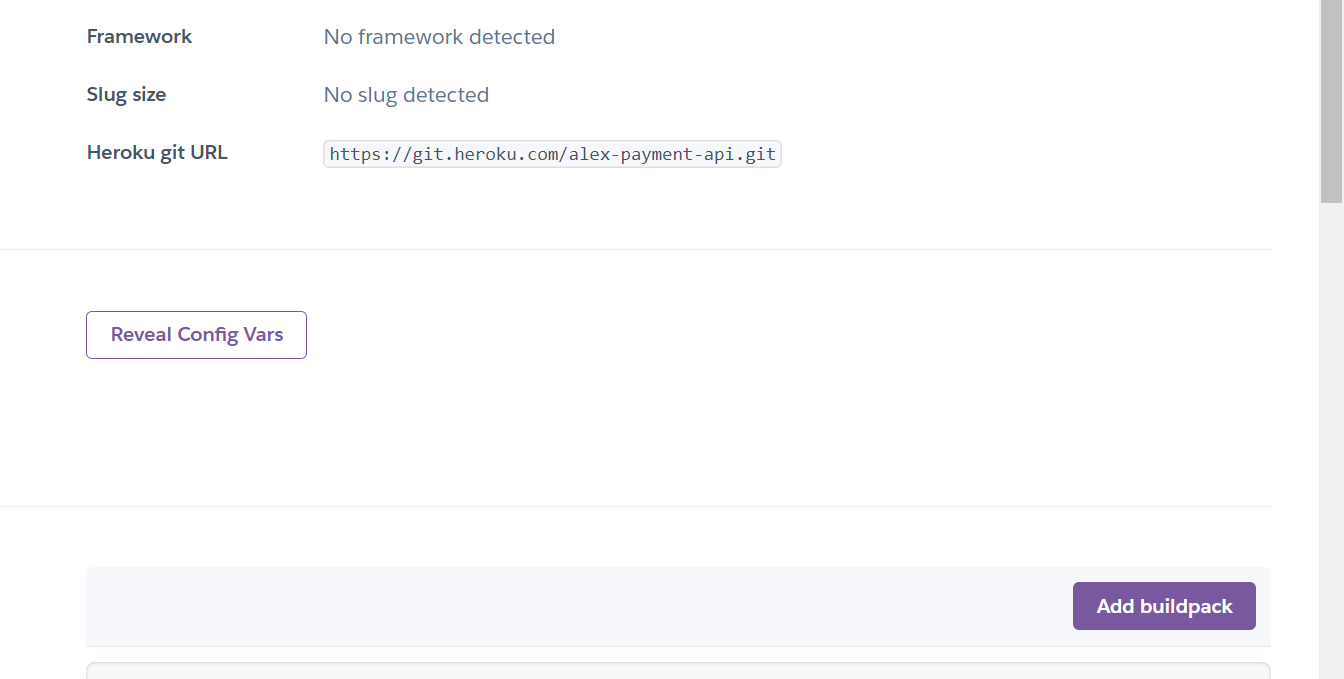
Buka Heroku, dan create new app



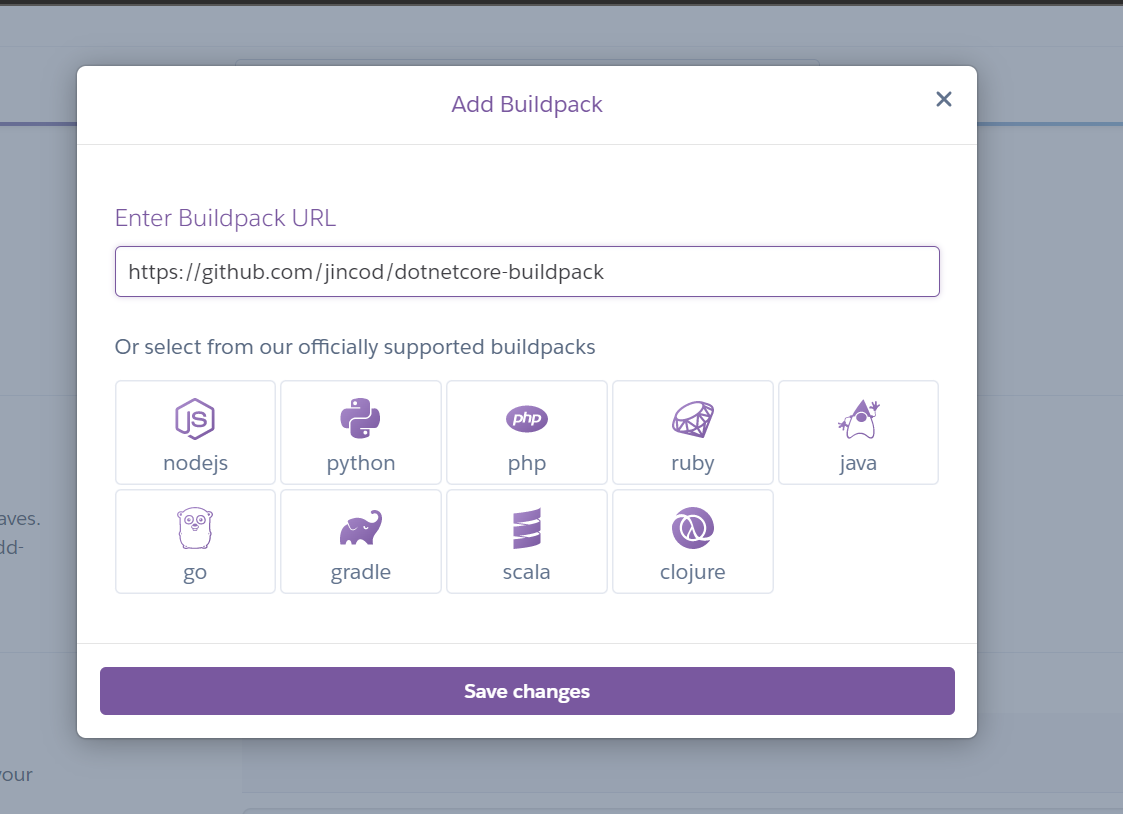
Masukkan nama, dan klik tombol create app



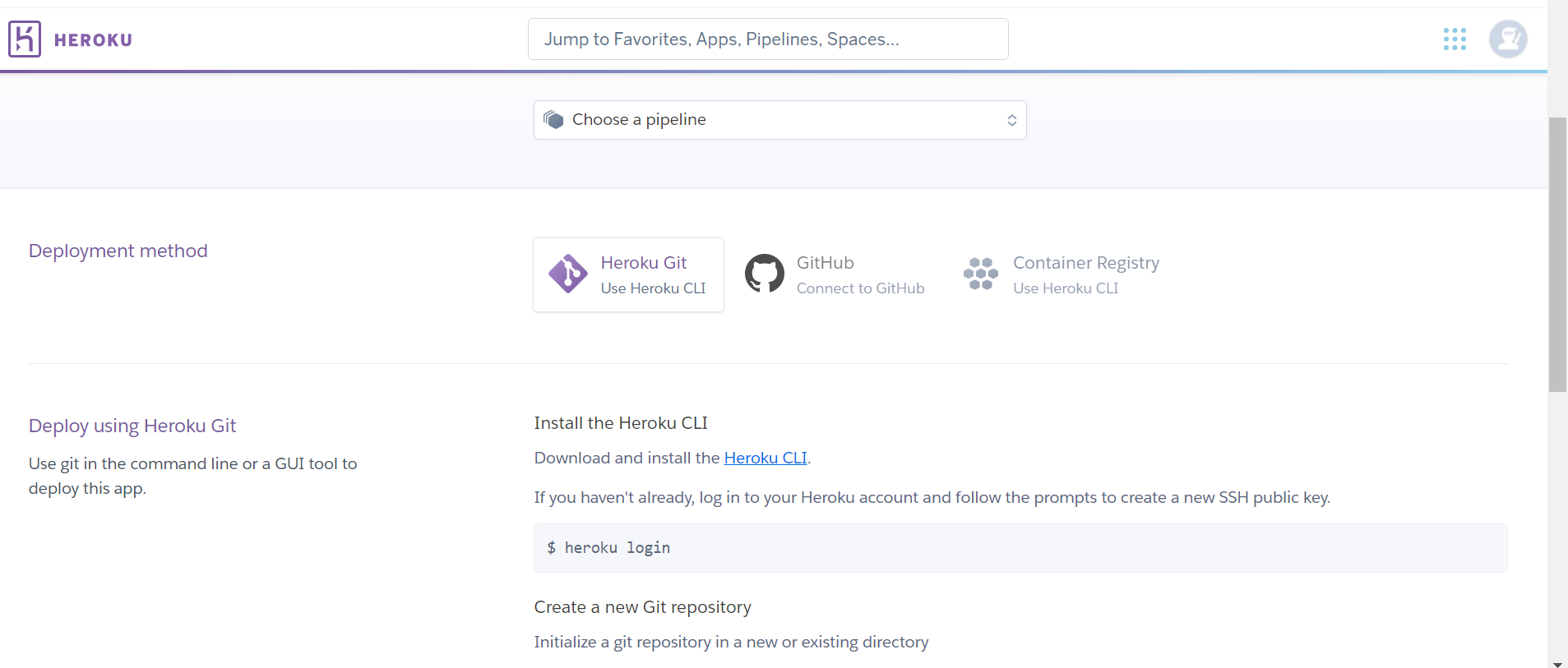
Masuk ke menu settings, dan klik tombol add buildpack



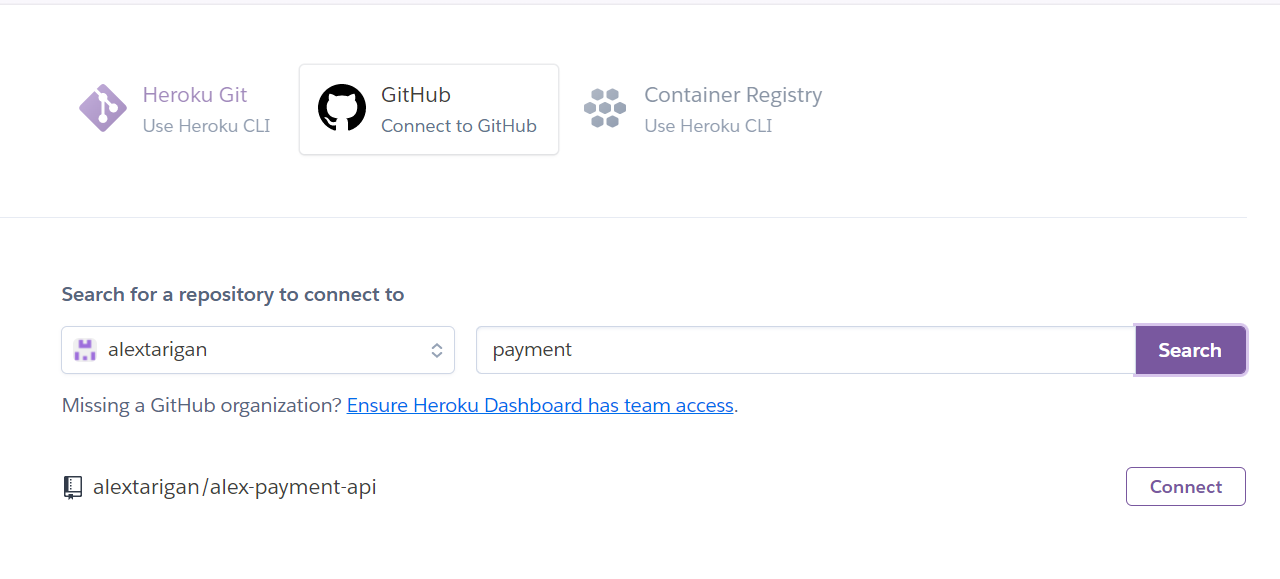
Masukkan link jincod, lalu klik tombol save change



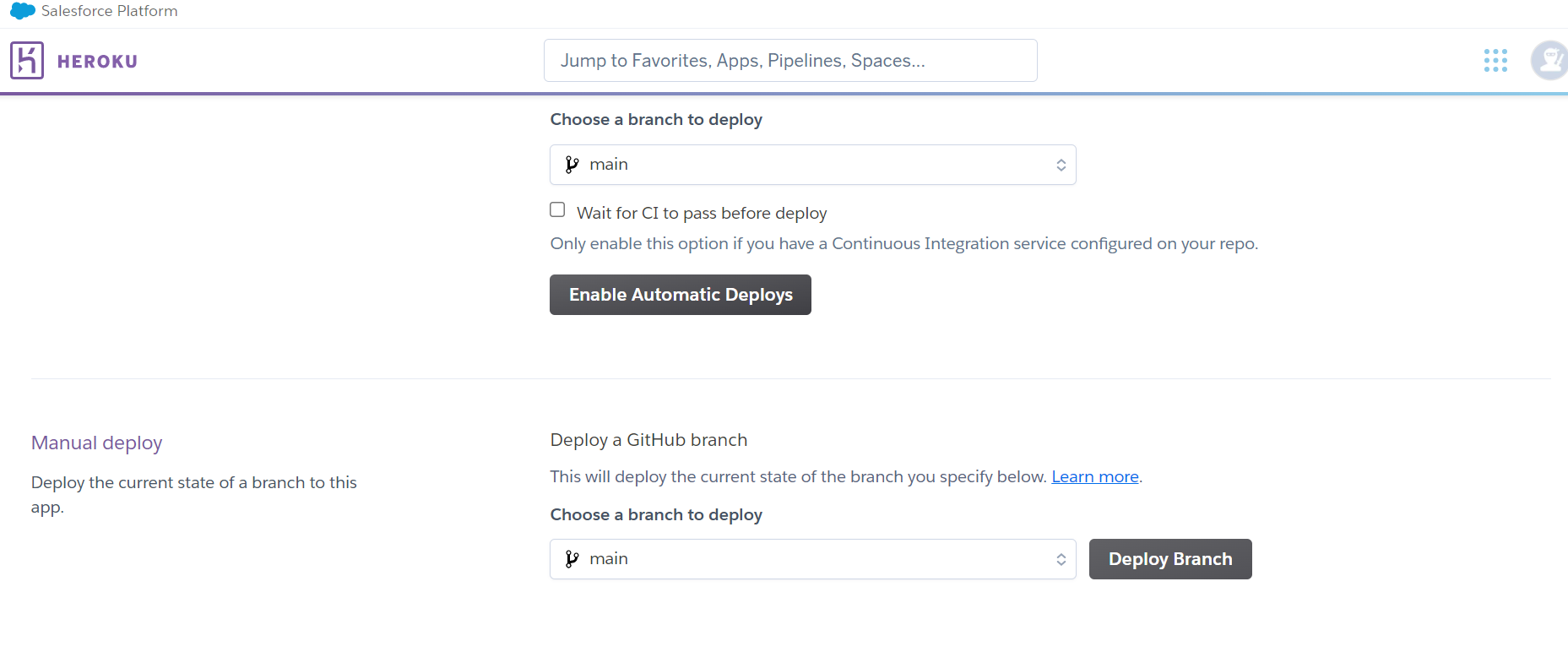
Masuk ke menu deploy, lalu connect to github



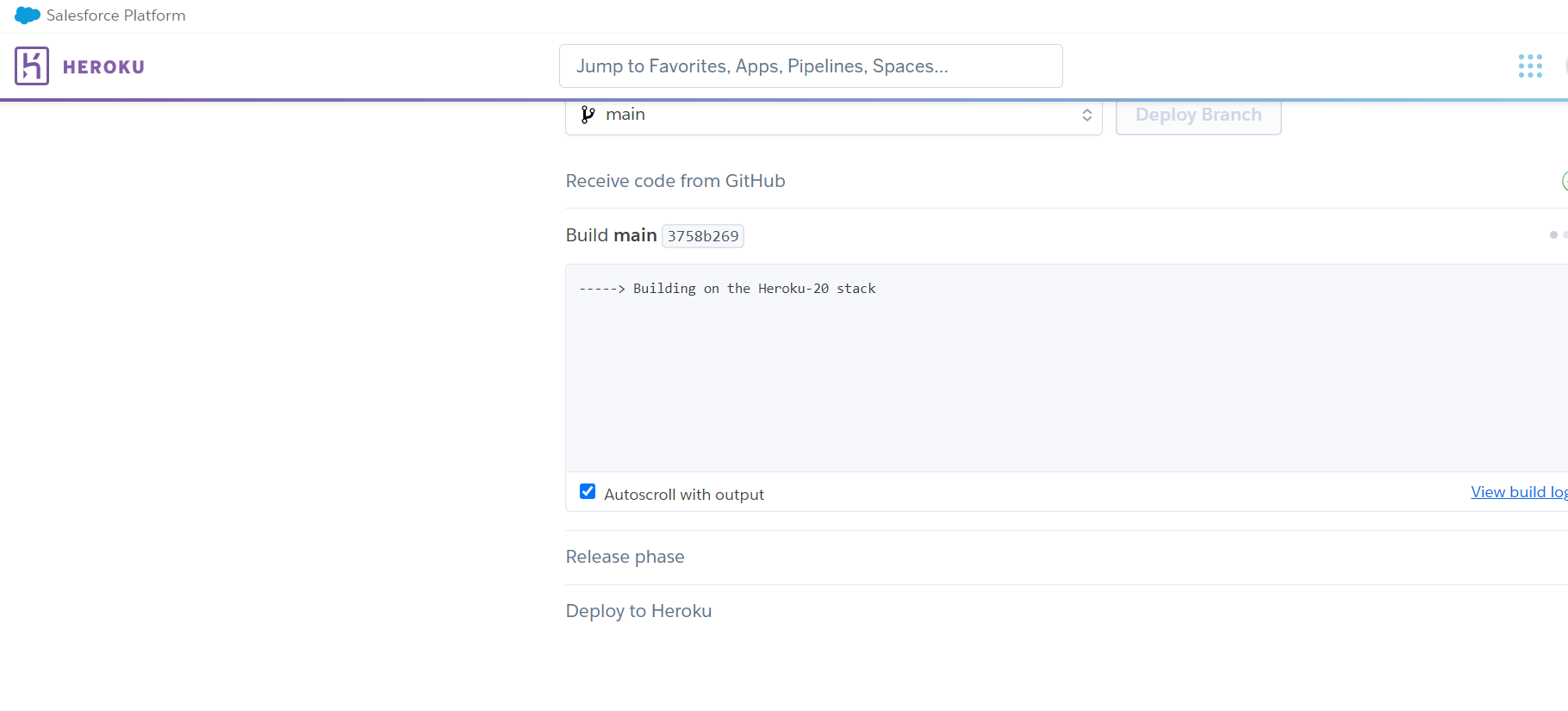
Cari repo yang telah kita buat tadi, lalu klik connect



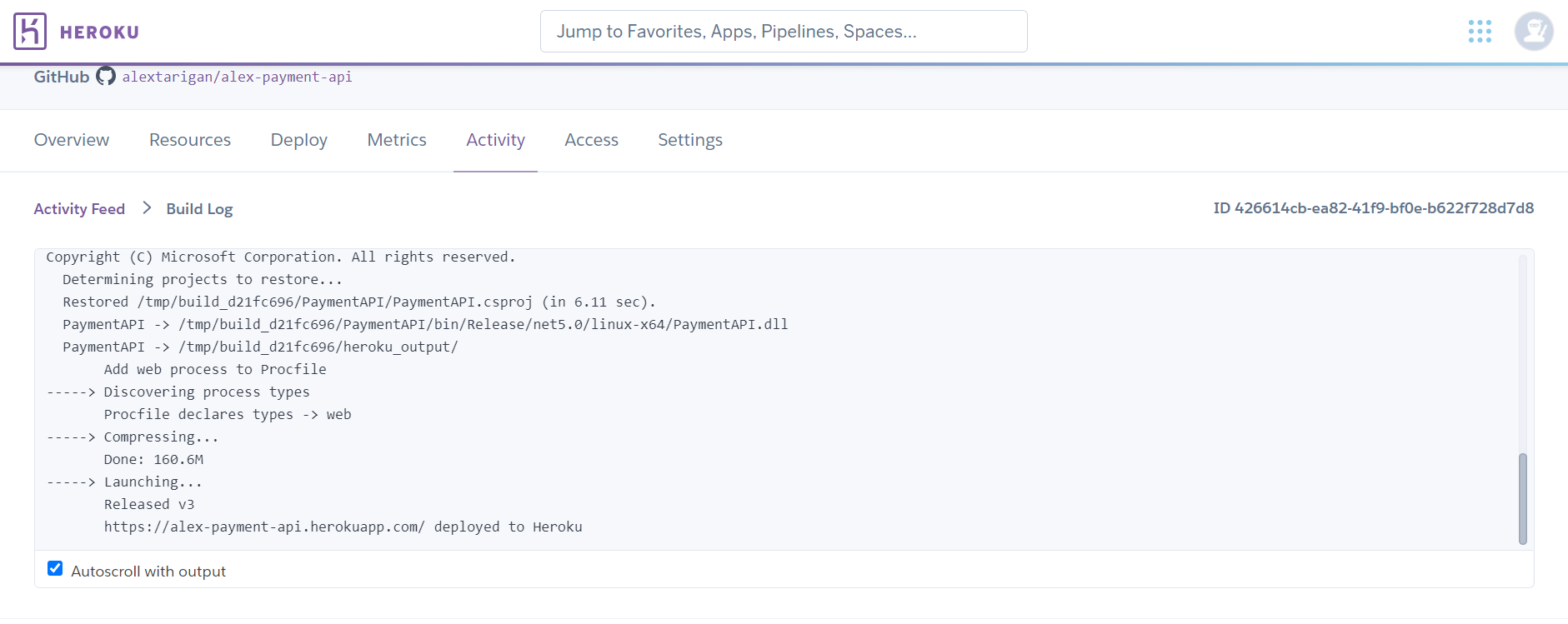
Klik tombol deploy branch



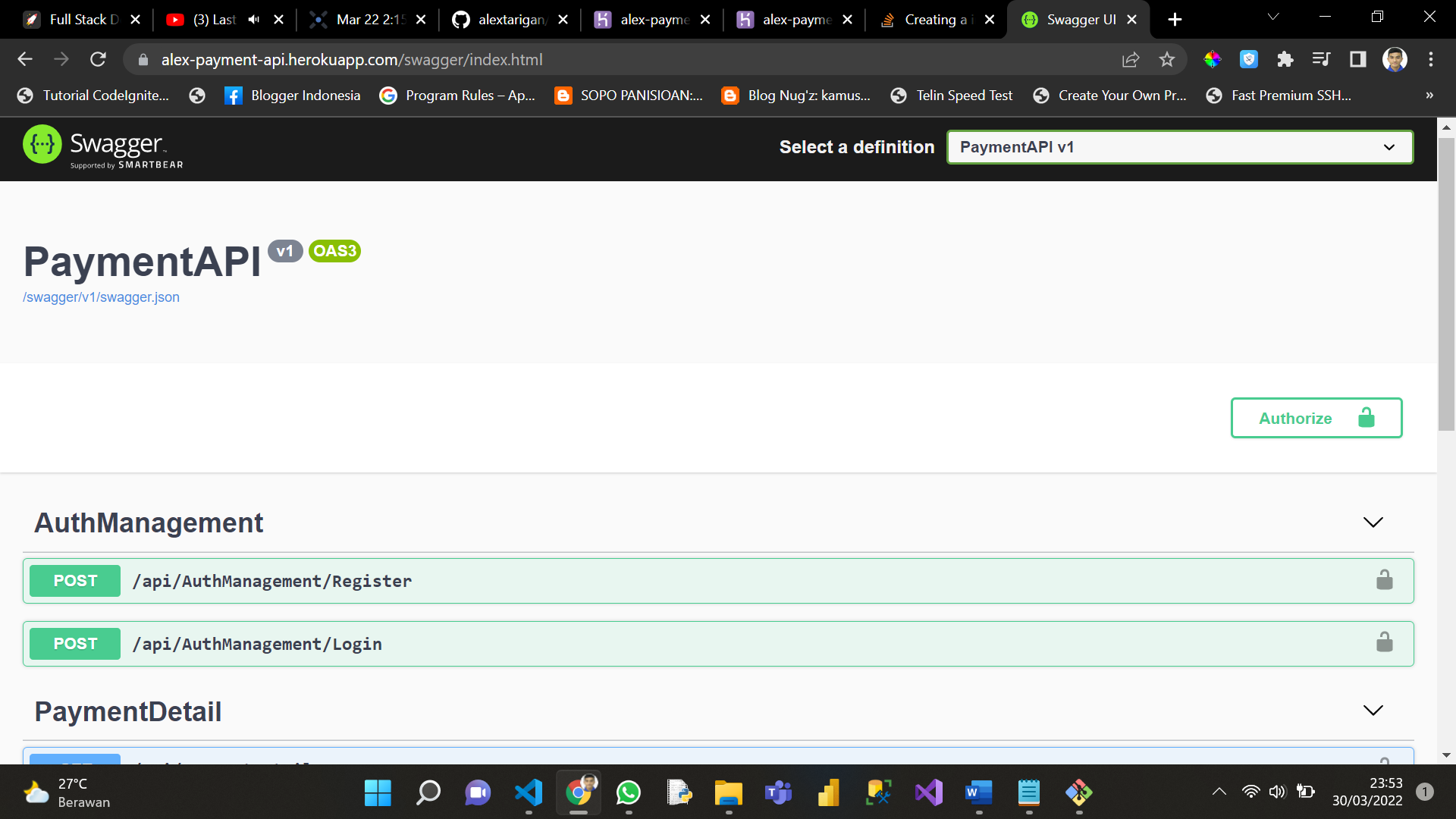
Tunggu sampai proses deploy selesai



Deploy telah selesai, ditandai dengan muncullnya link app Heroku yang telah kita buat

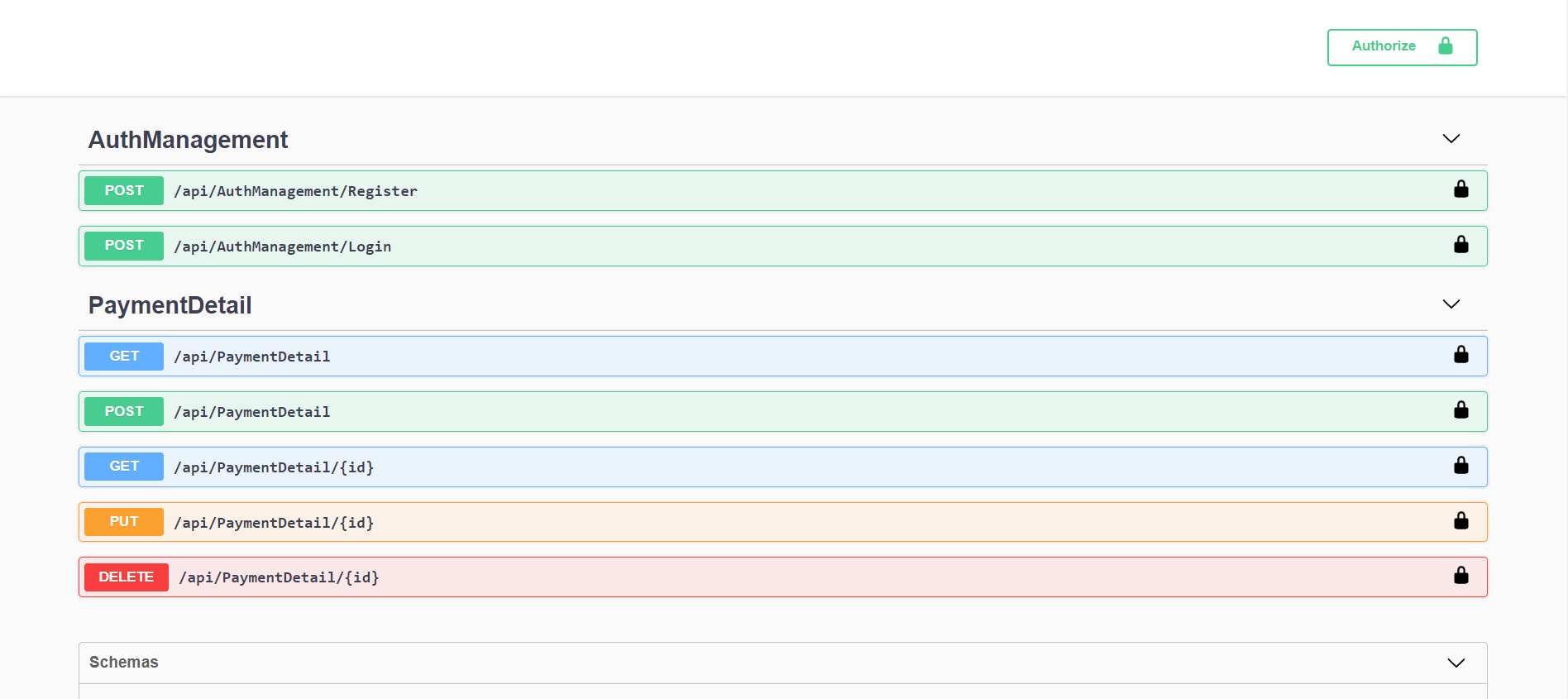


Buka <https://alex-payment-api.herokuapp.com/swagger/index.html>



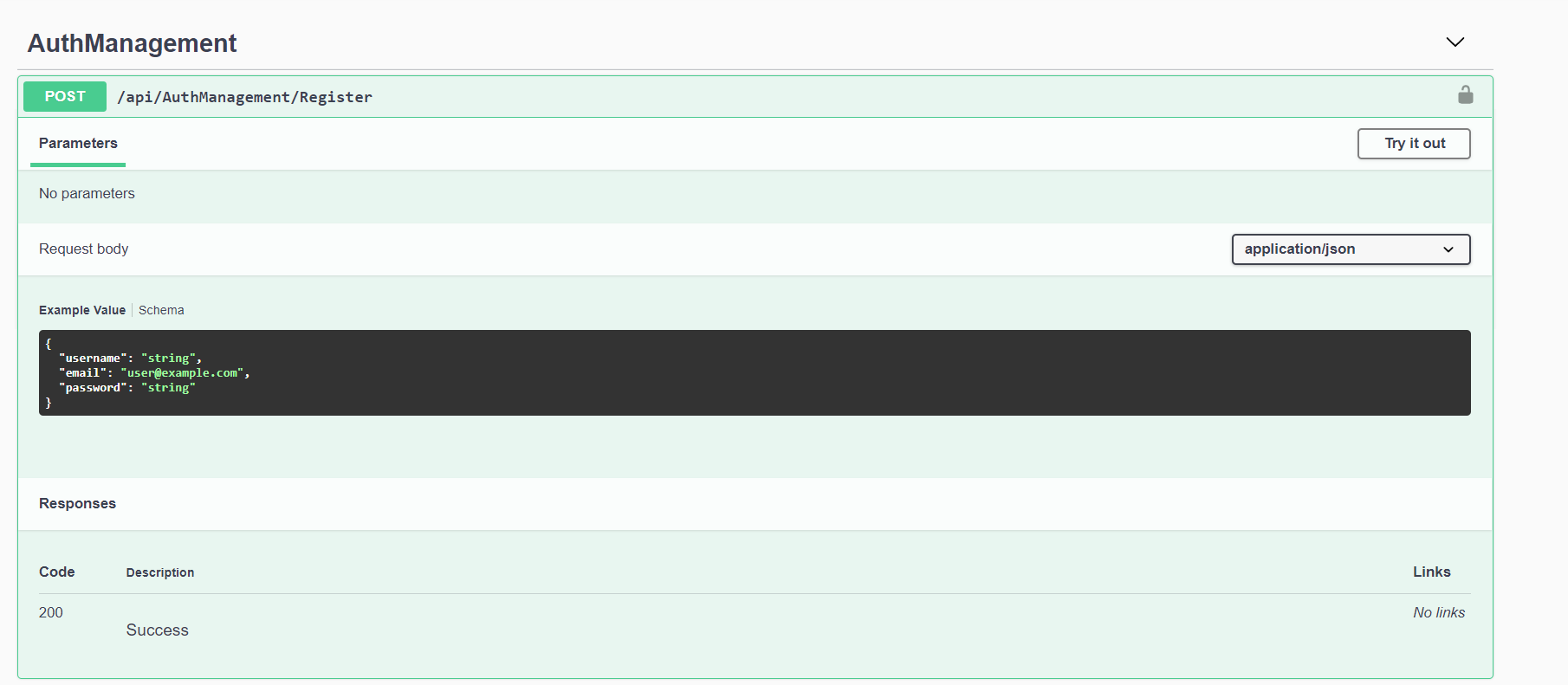
Pengetesan Di swagger dan Postman

1. Swagger

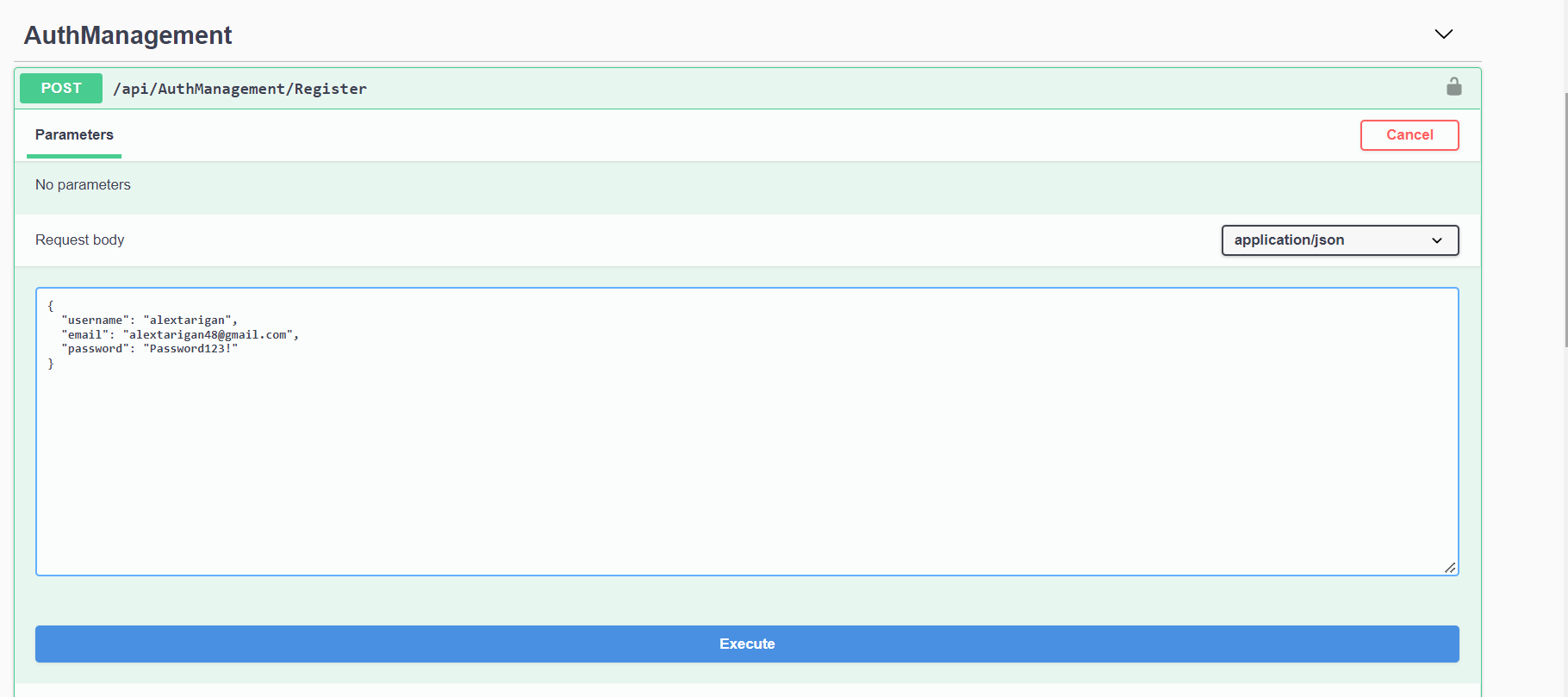


1. Register

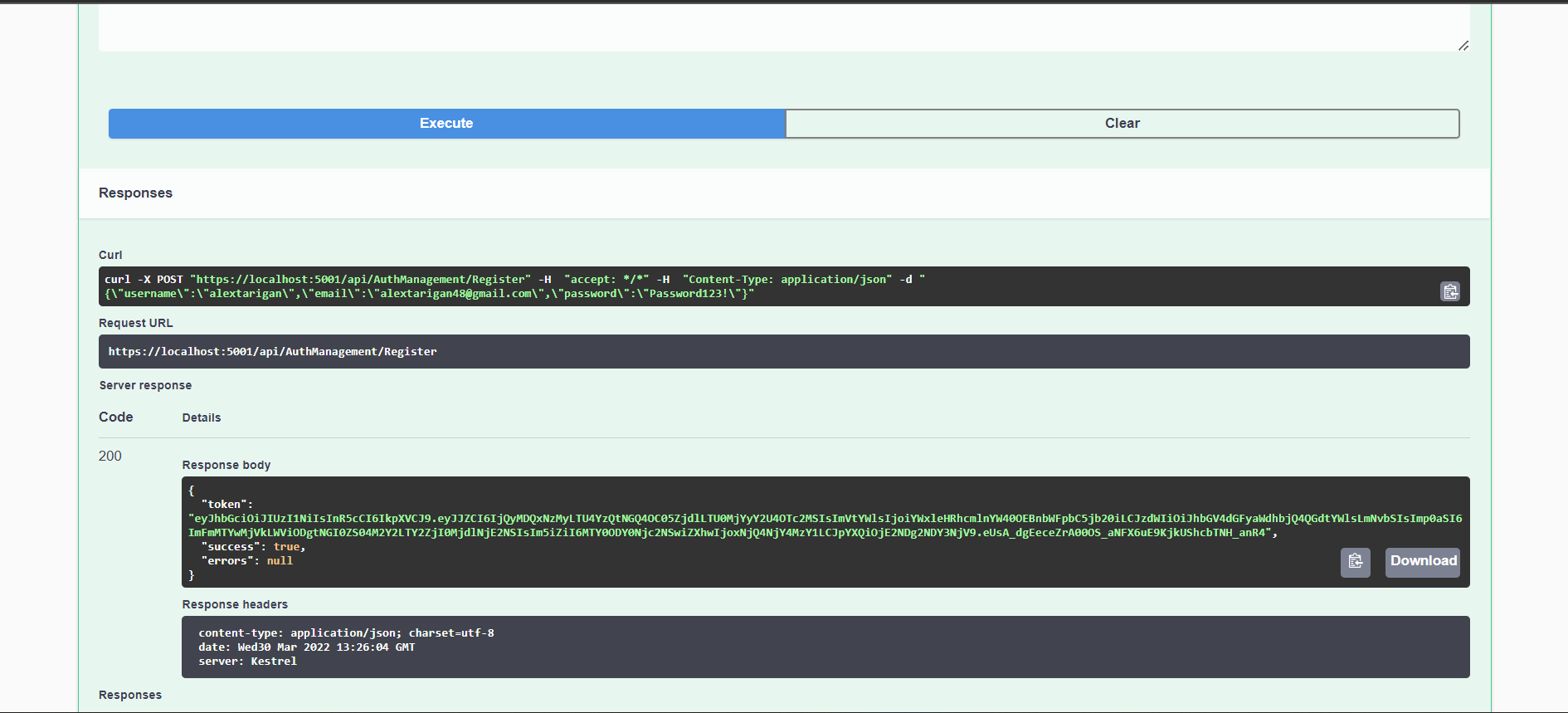
Membuat sebuah akun menggunakan controller Authmanagement, pertama klik try it out



Lalu masukkan username, email dan password yang ingin dibuat.

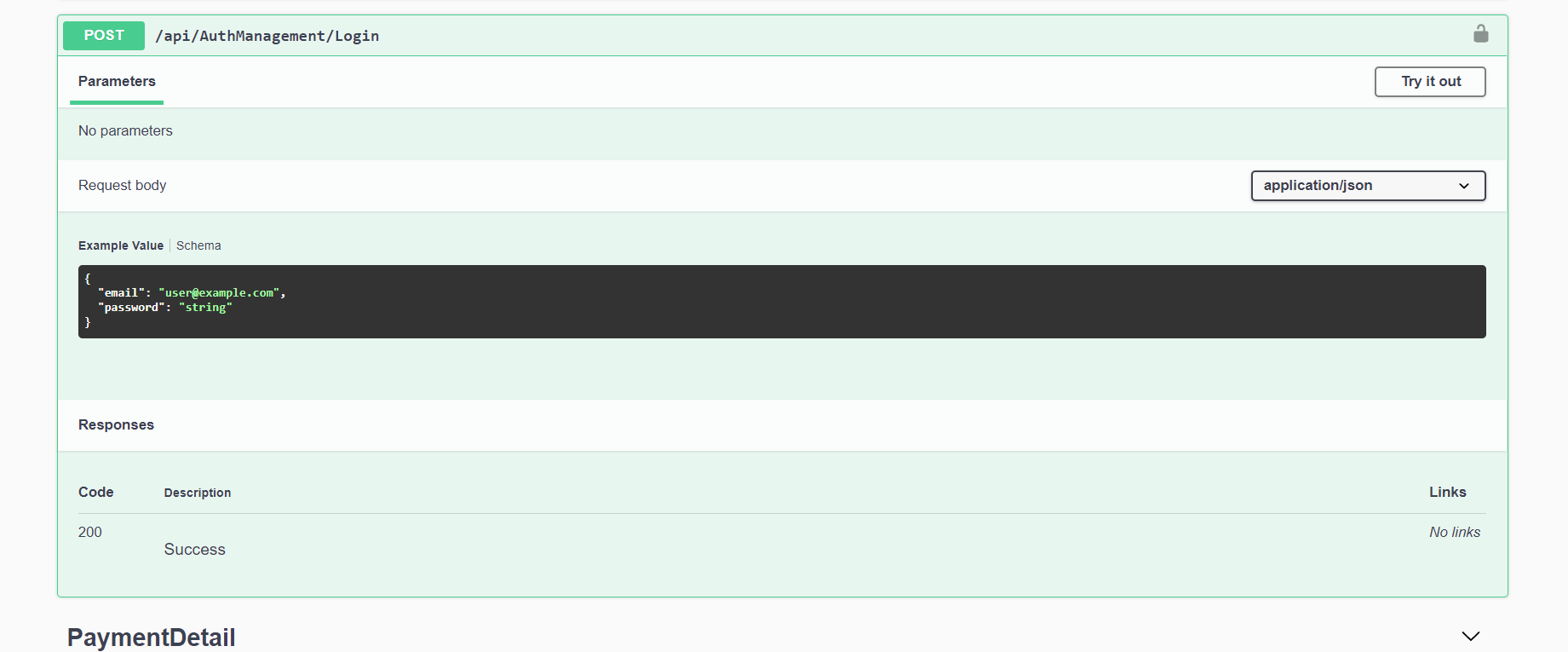


Akun berhasil digunakan, dan swagger memberi response success true, dan menghasilkan token.



2. Login

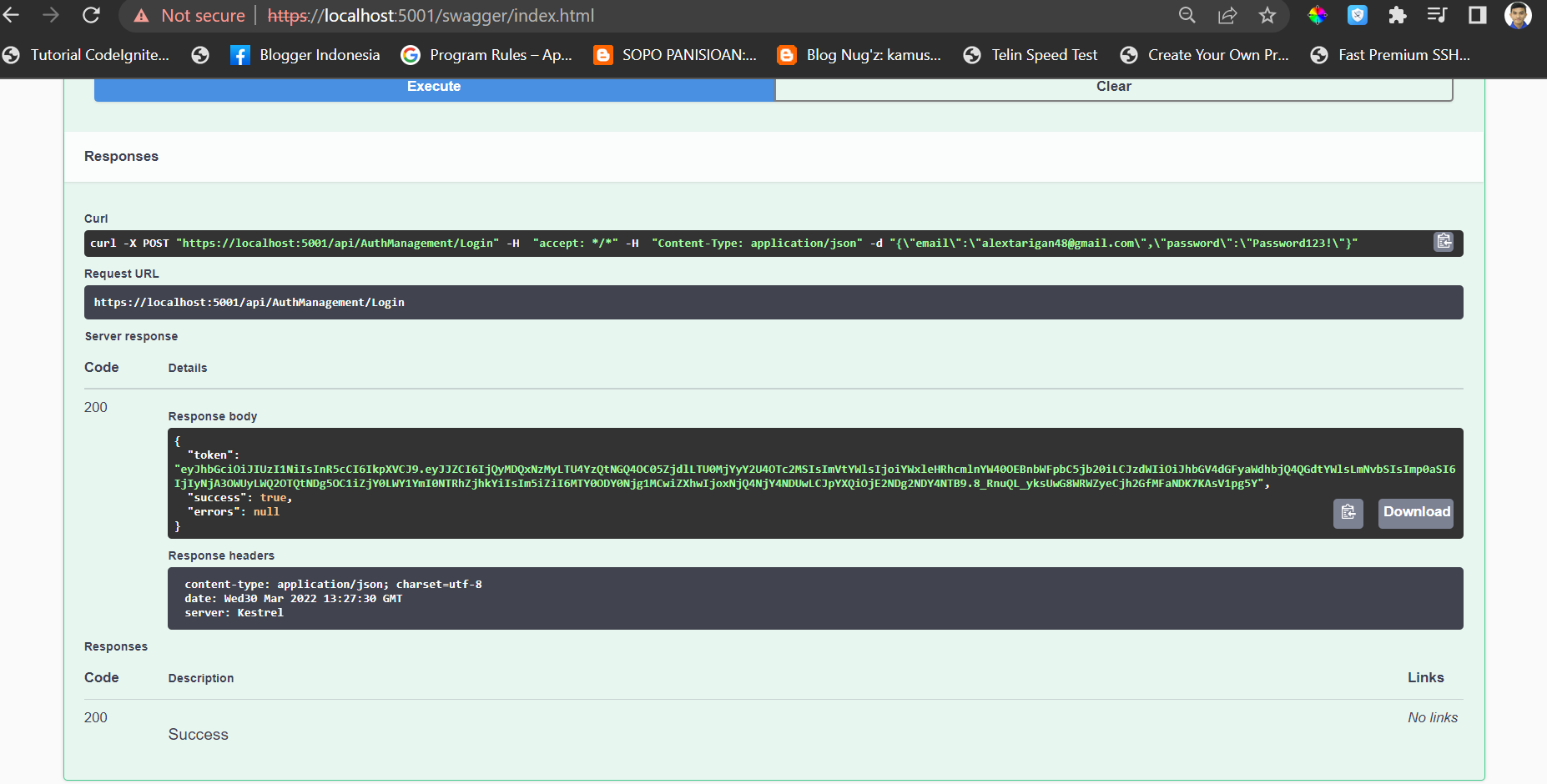
Login akun menggunakan controller Authmanagement, pertama klik try it out



Masukkan email dan password dengan benar

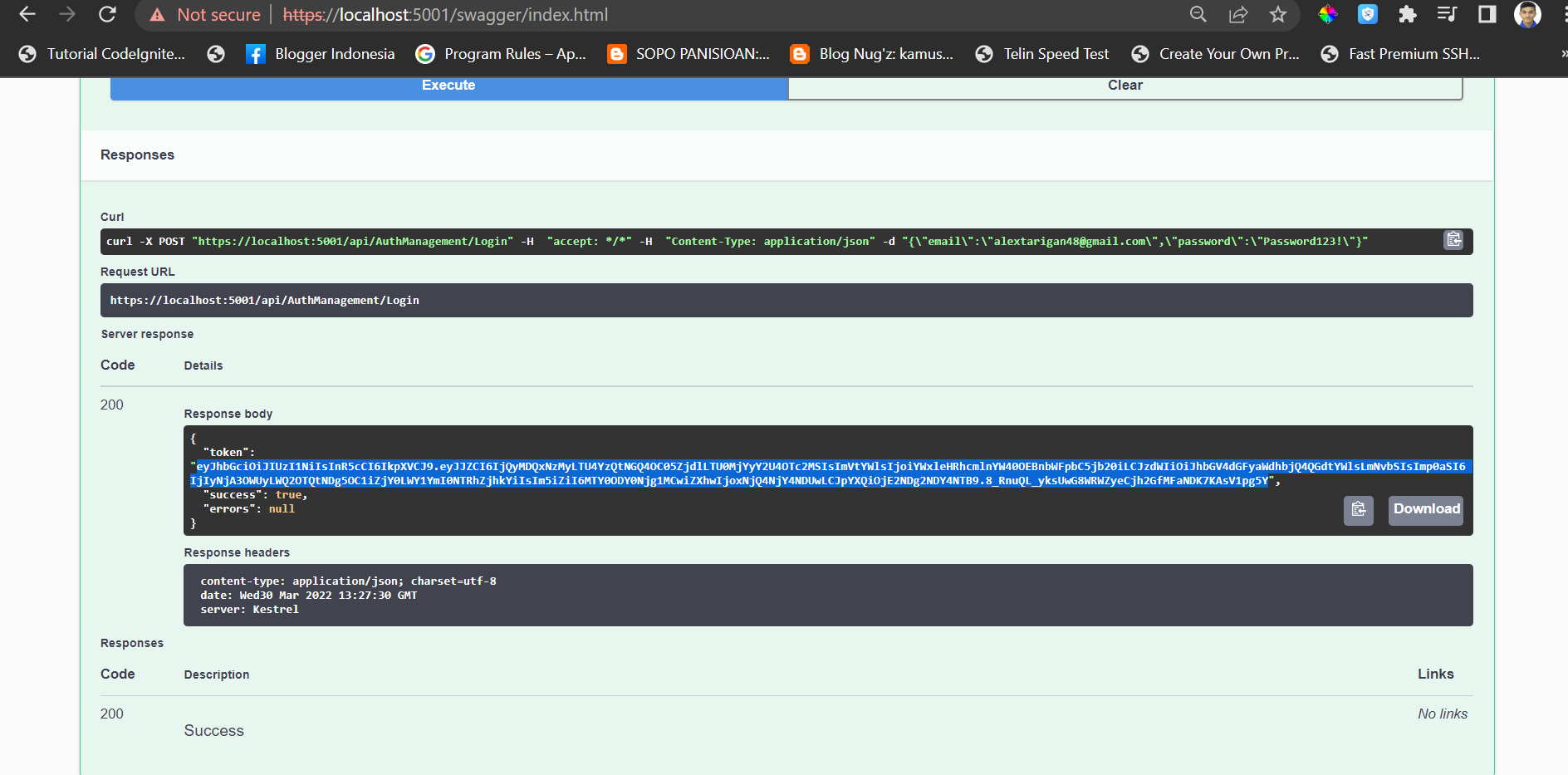


Swagger memberi response dan mereturn sebuah token.

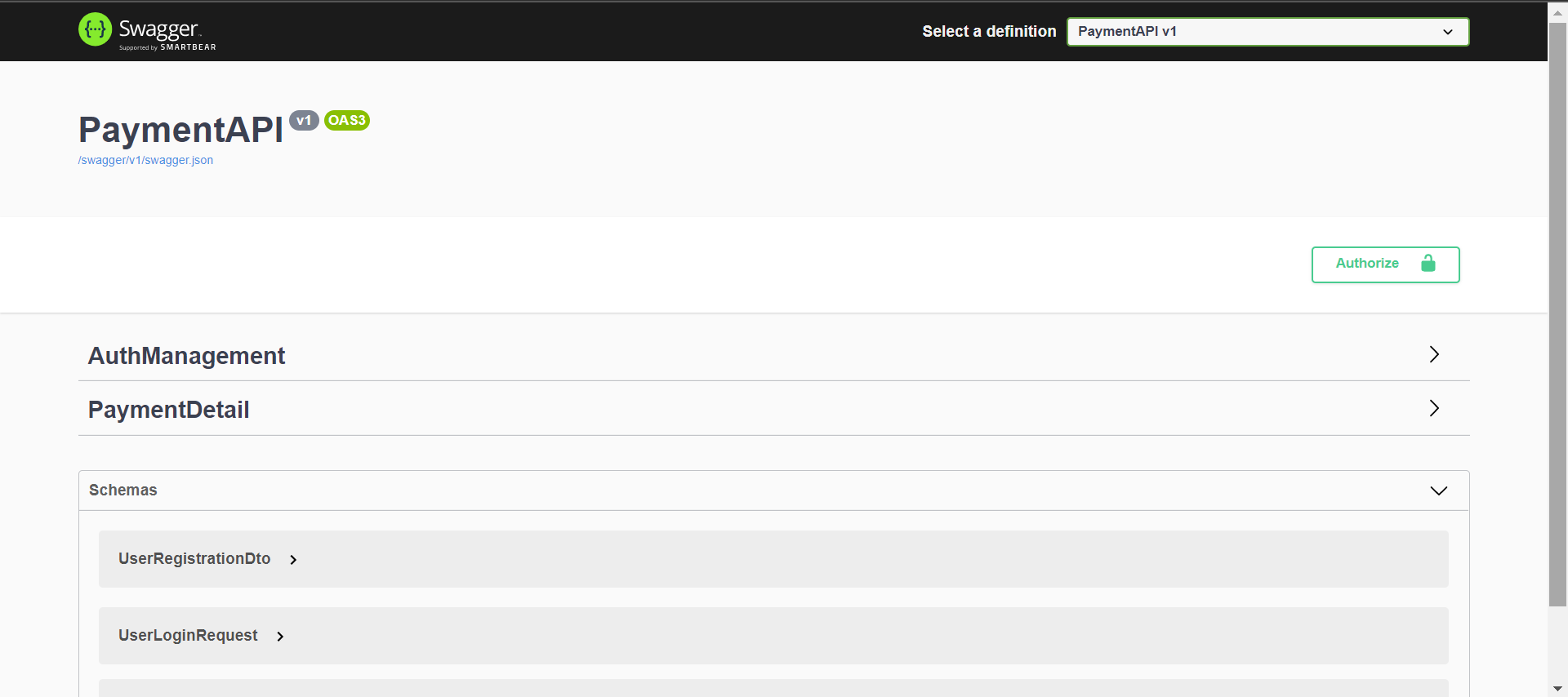


3. Bearer Token ( Authorize)

Salin token yang didapatkan setelah berhasil login



Klik tombol Authorize

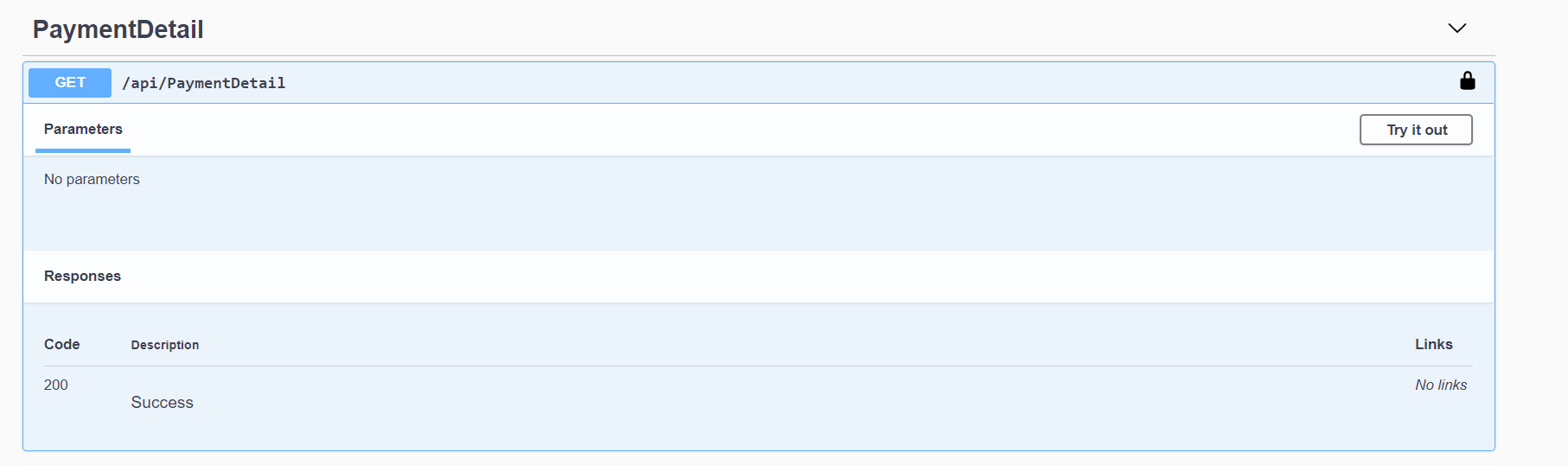


Masukkan token yang telah didapatkan tadi, lalu klik tombol authorize agar dapat menggunakan controller paymentDetails

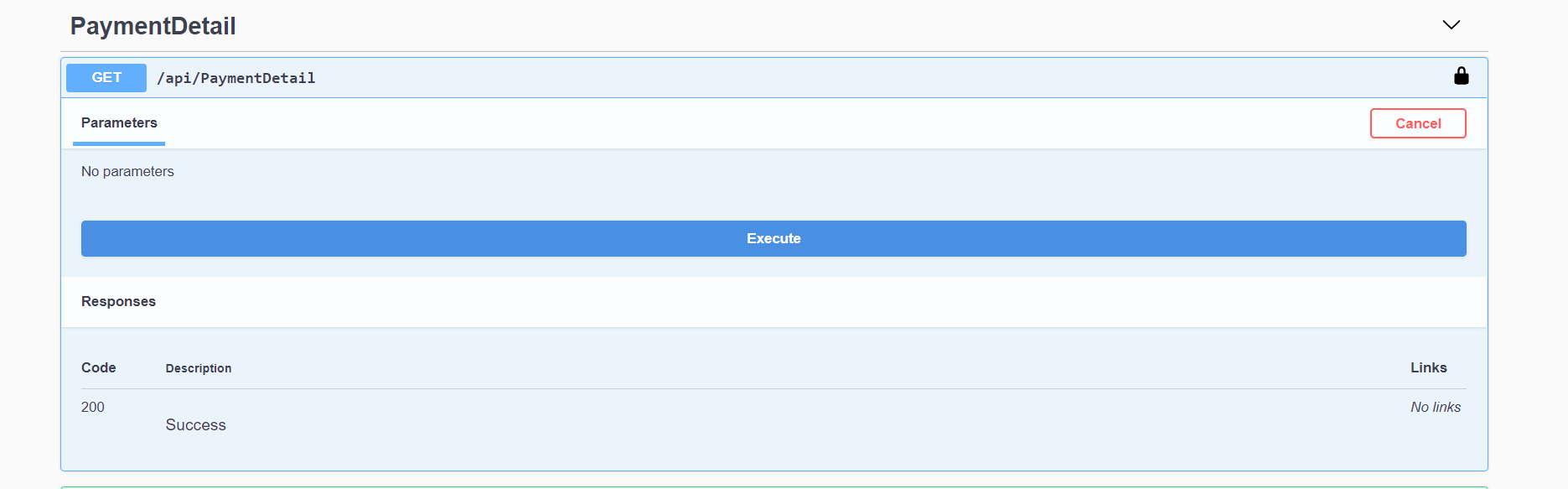


4. Get all Data

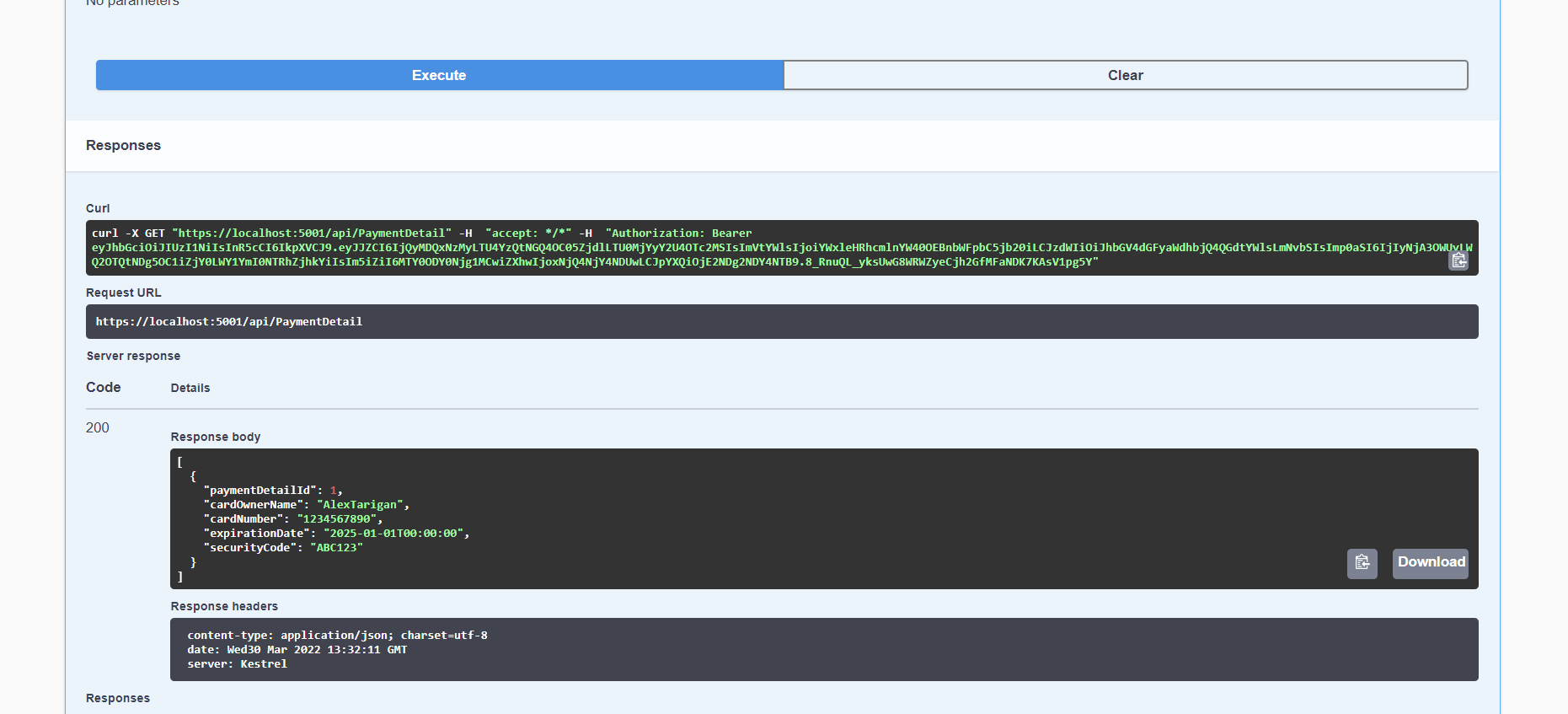
Klik Try it out



Klik tombol execute

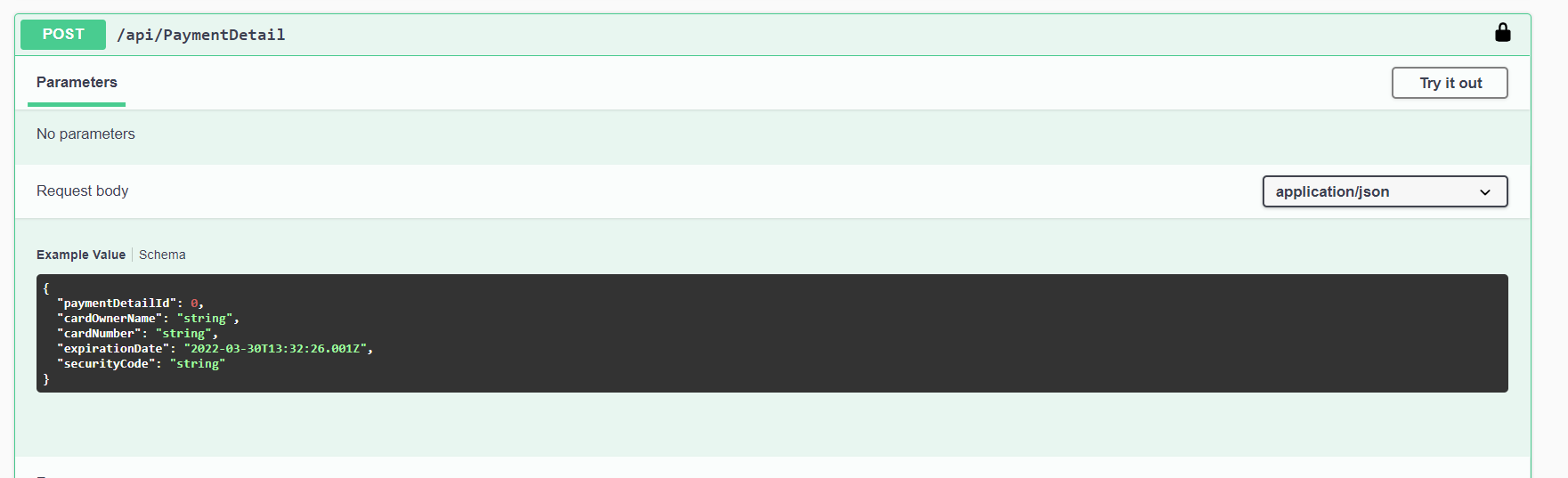


Swagger memberikan response berupa semua isi yang ada pada table payments, dikarenakan tadi hanya menginsert 1 data, makan hanya 1 data yang ditampilkan.

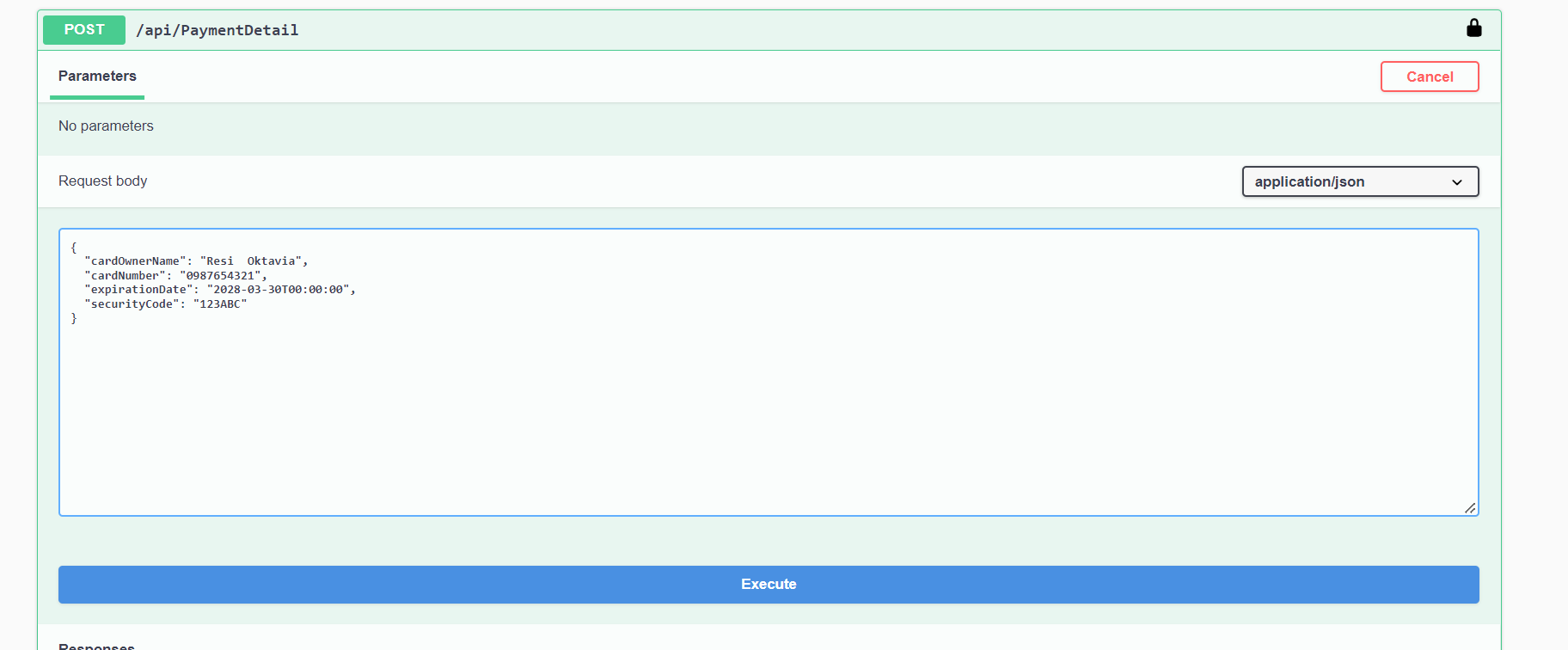


5. Post Data

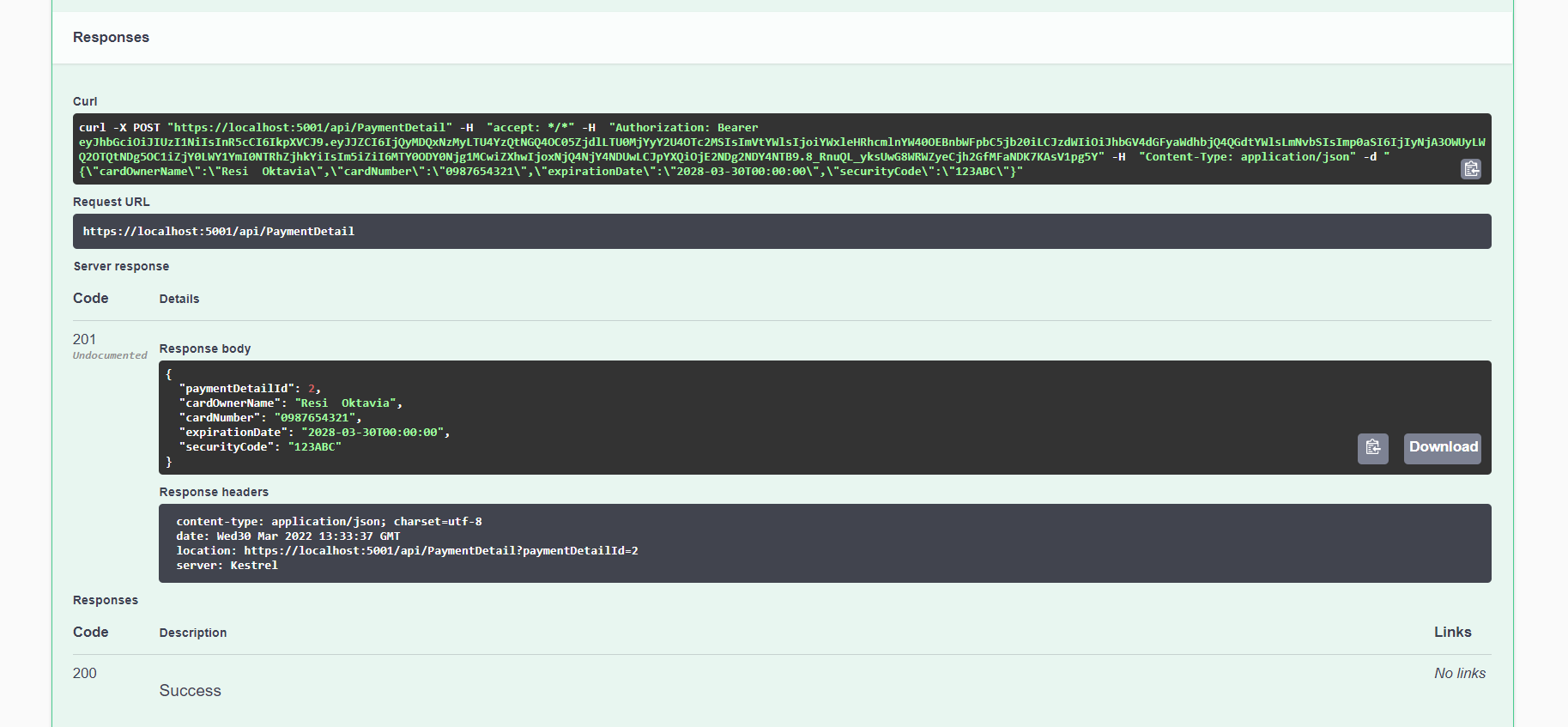
Klik tombol try it out



Masukkan data sesuai kolom yang tersedia. Paymentdetailid tidak perlu dimasukkan, karena kolom paymentdetailid merupakan kolom yang bersifat auto increment.

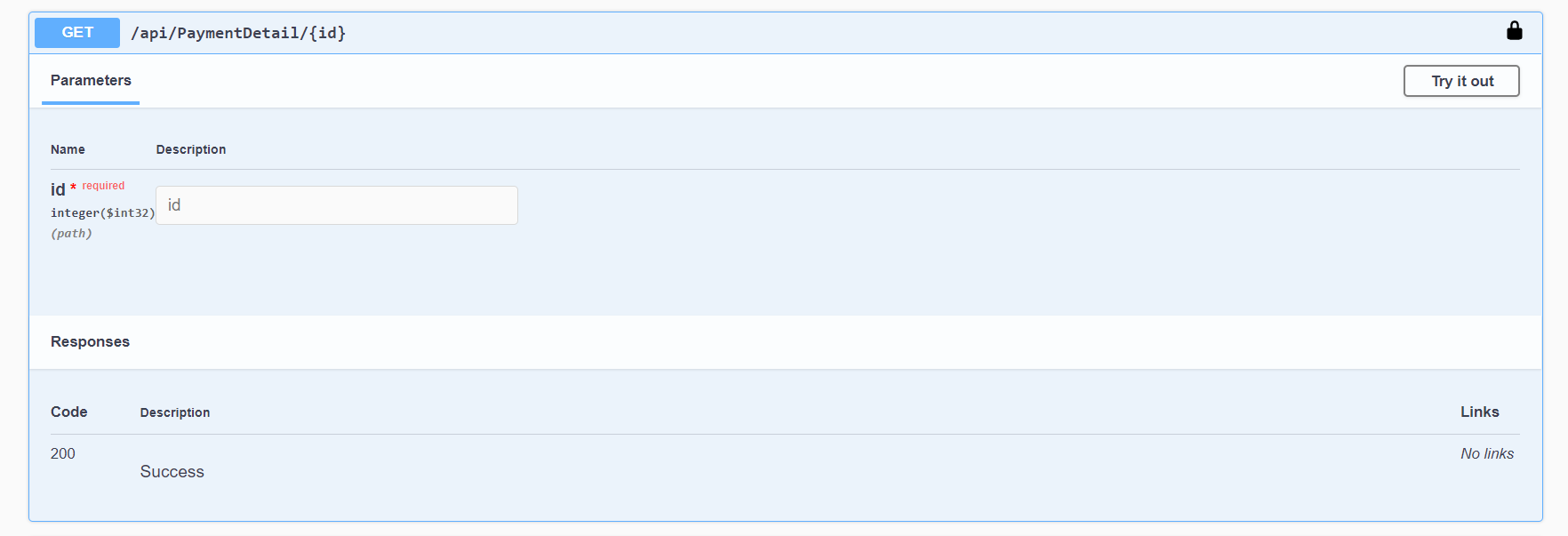


Data berhasil dimasukkan ke table payments

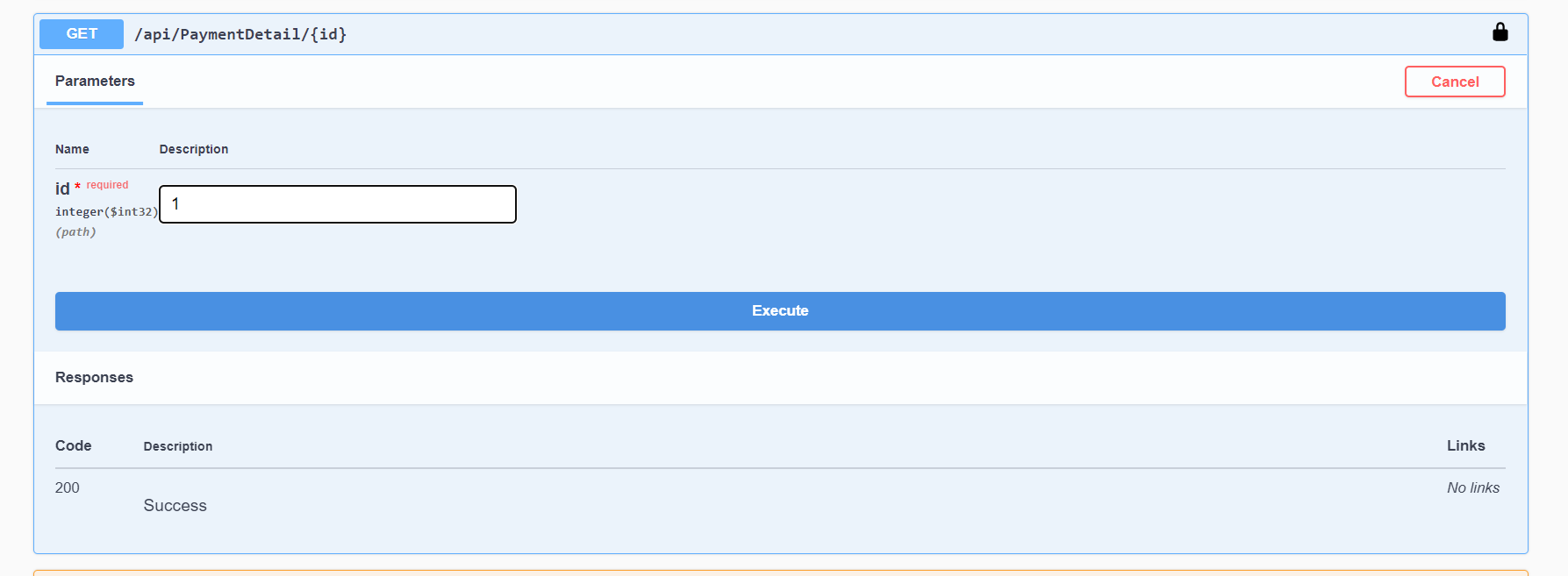


6. Get data By Id

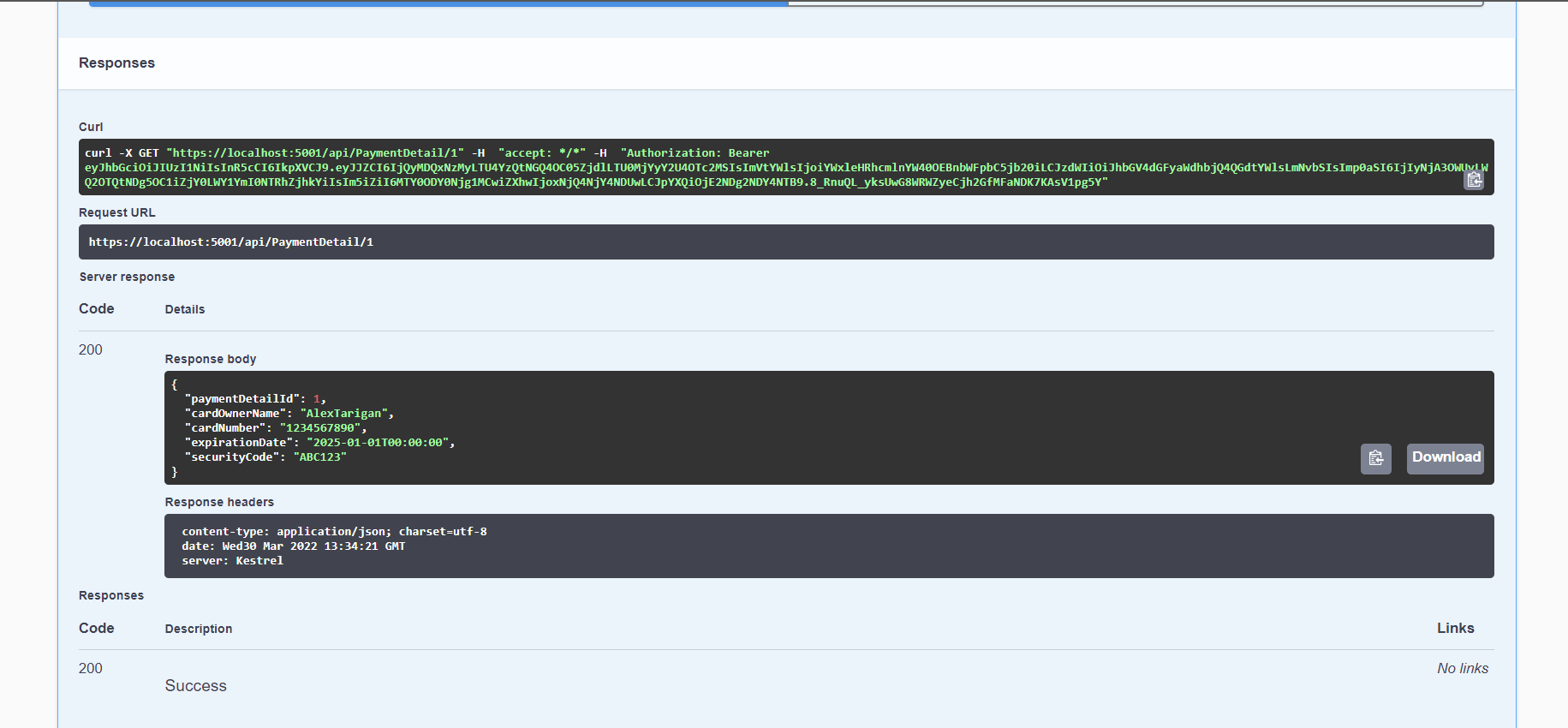
Klik tombol try it out



Masukkan paymentdetailid yang ingin ditampilkan

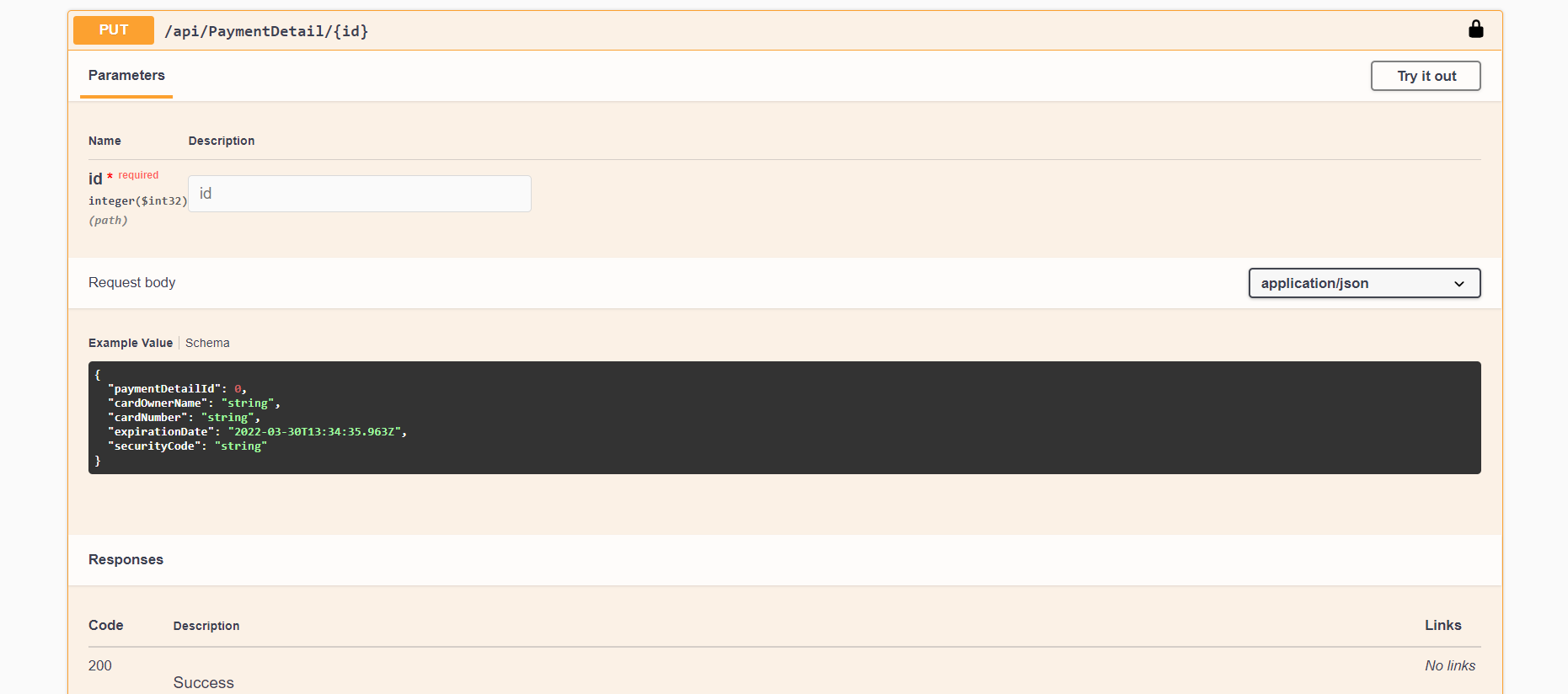


data berhasil ditampilkan

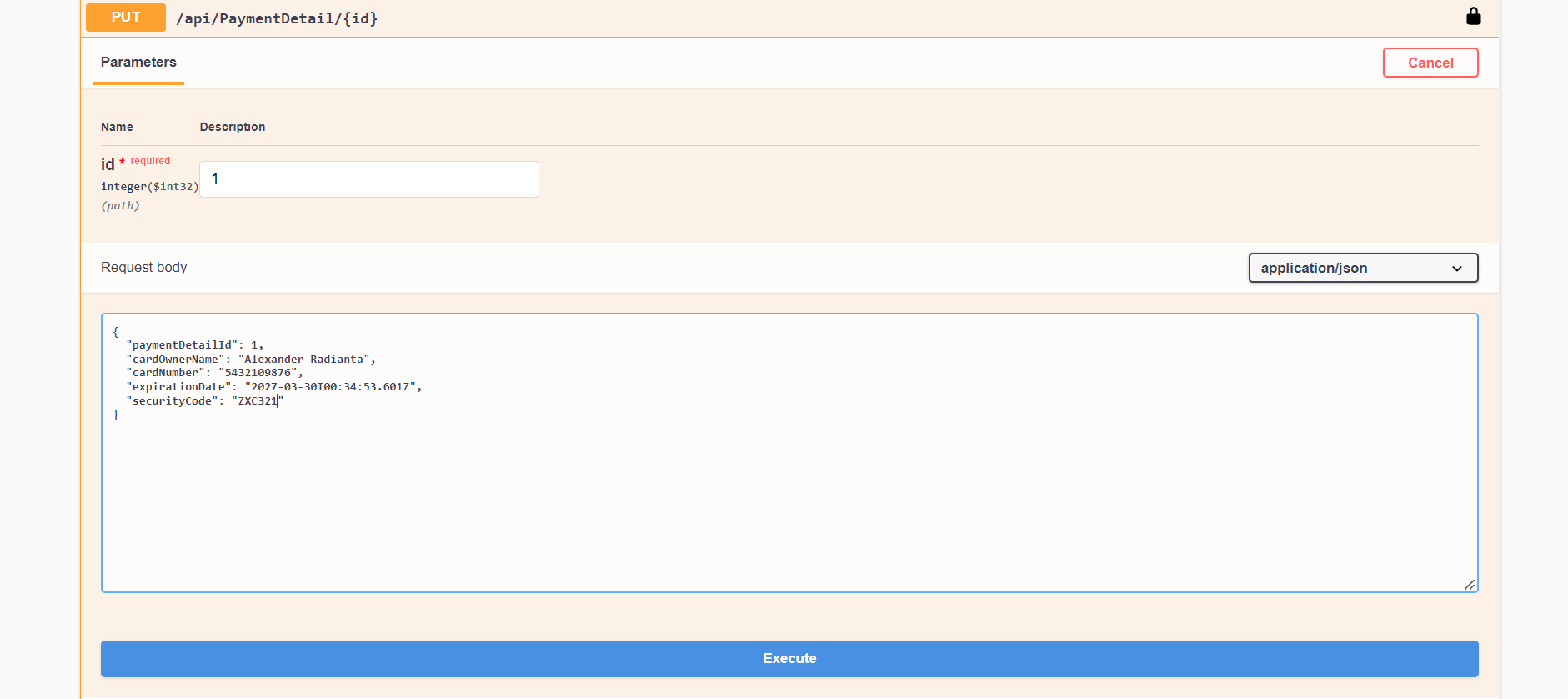


7. Put (edit) data

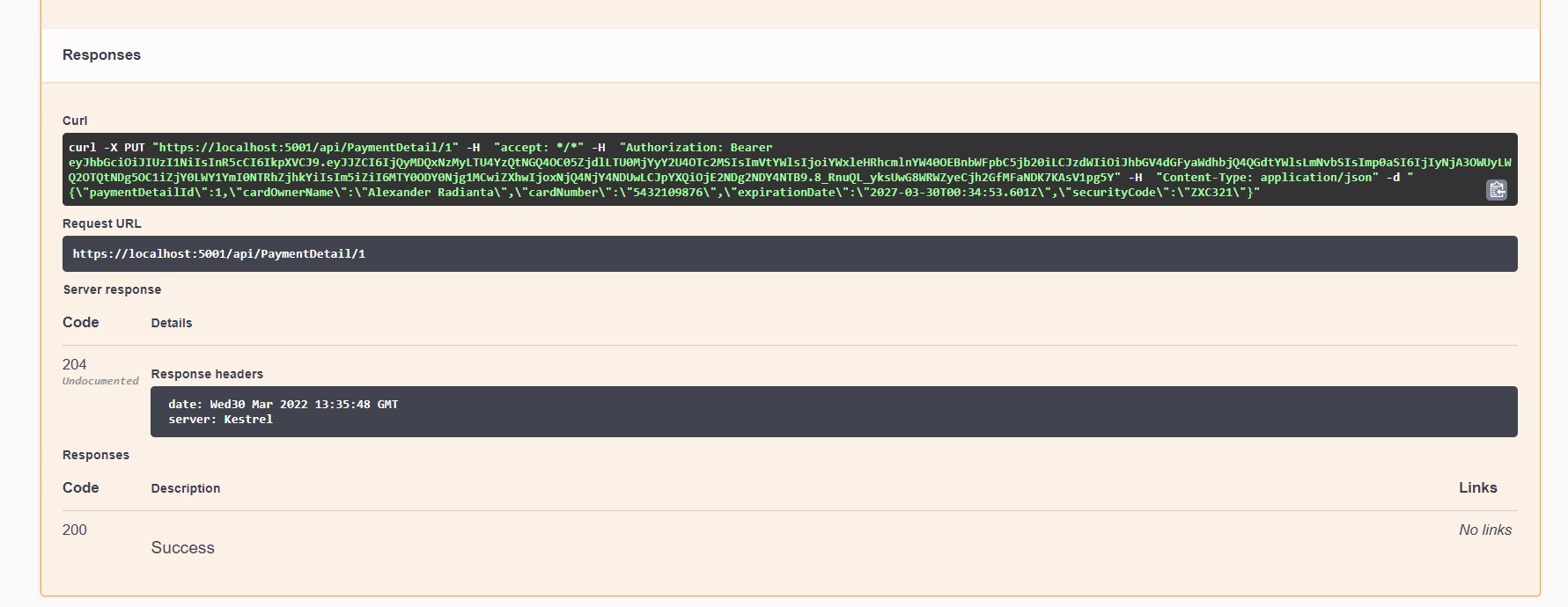
Klik tombol try it out



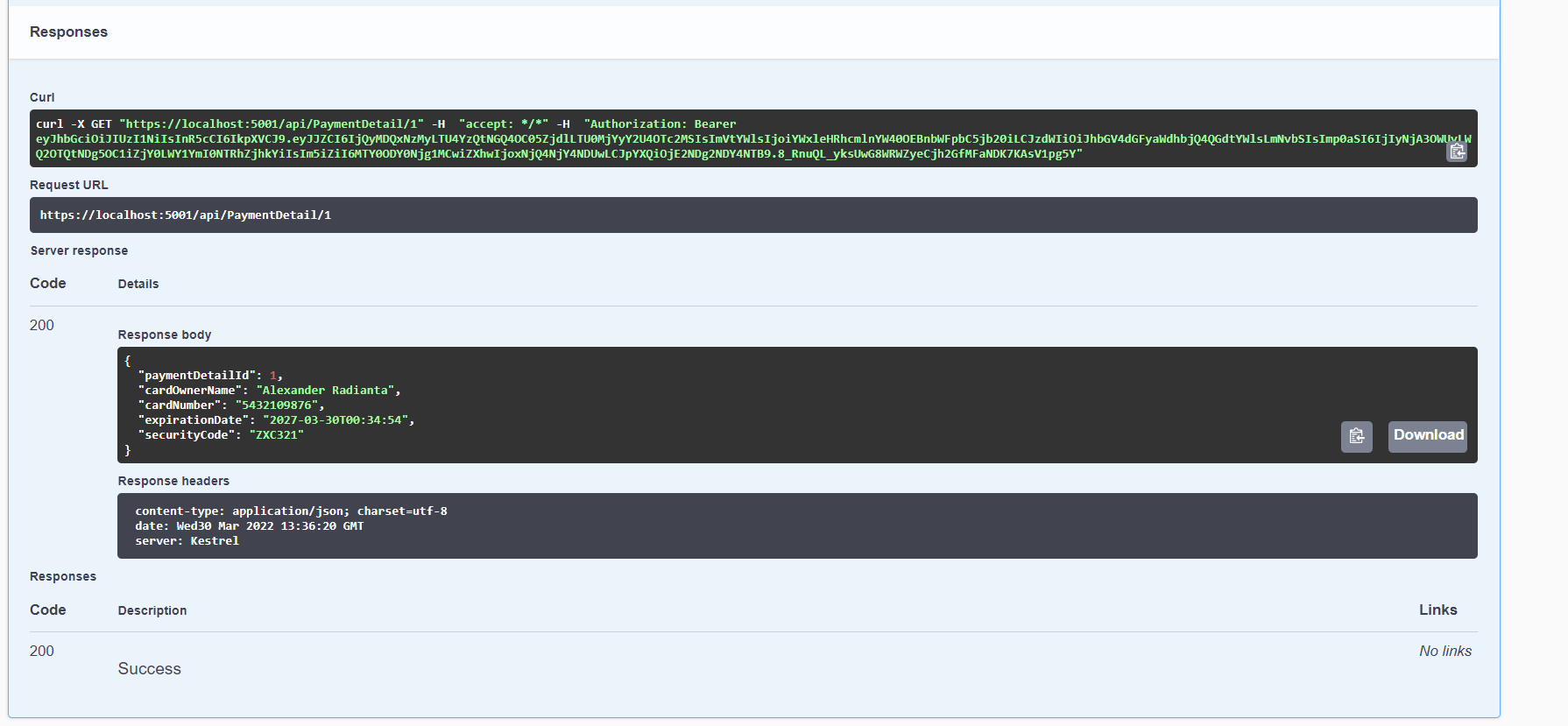
Masukkan paymentdetailid untuk data yang ingin diubah, lalu klik tombol execute



Swagger akan meresponse success

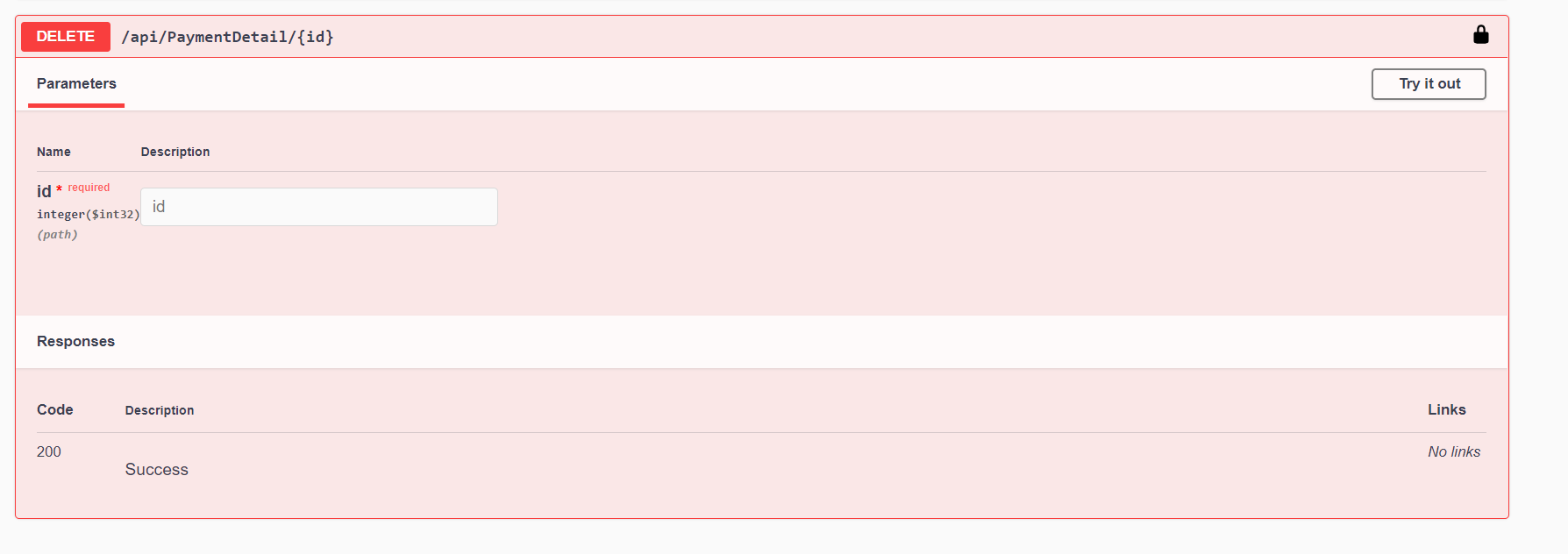


Data berhasil di ubah

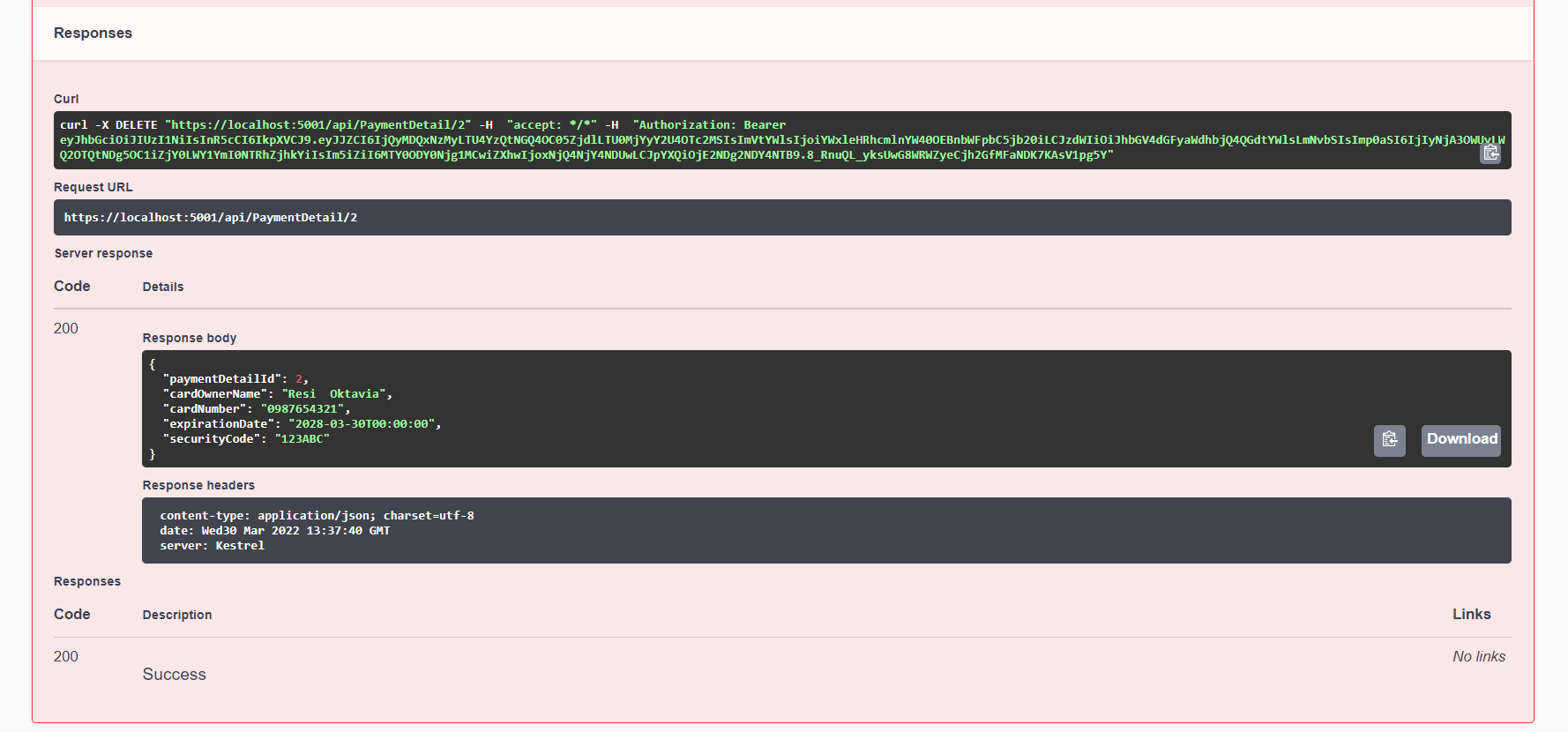


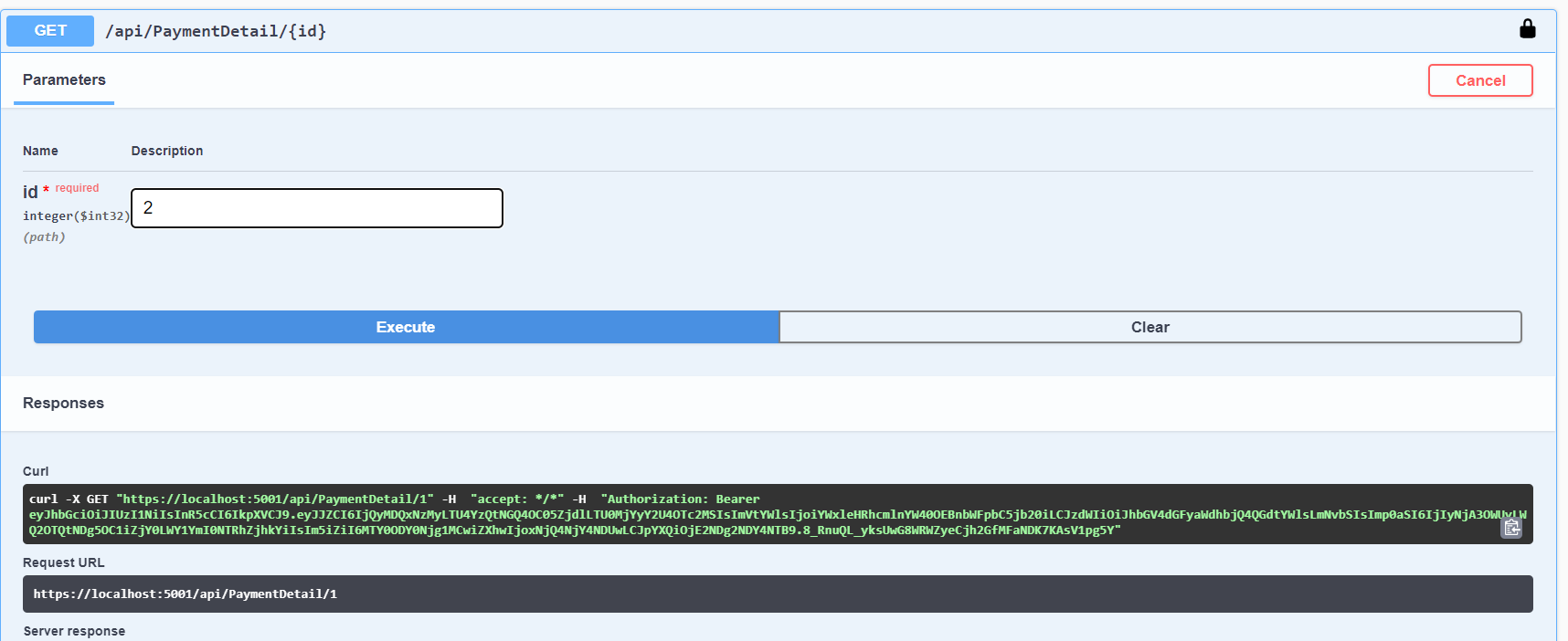
8. DELETE Data

Klik tombol try it out, lalu masukkan paymentdetailid , dan klik tombol execute

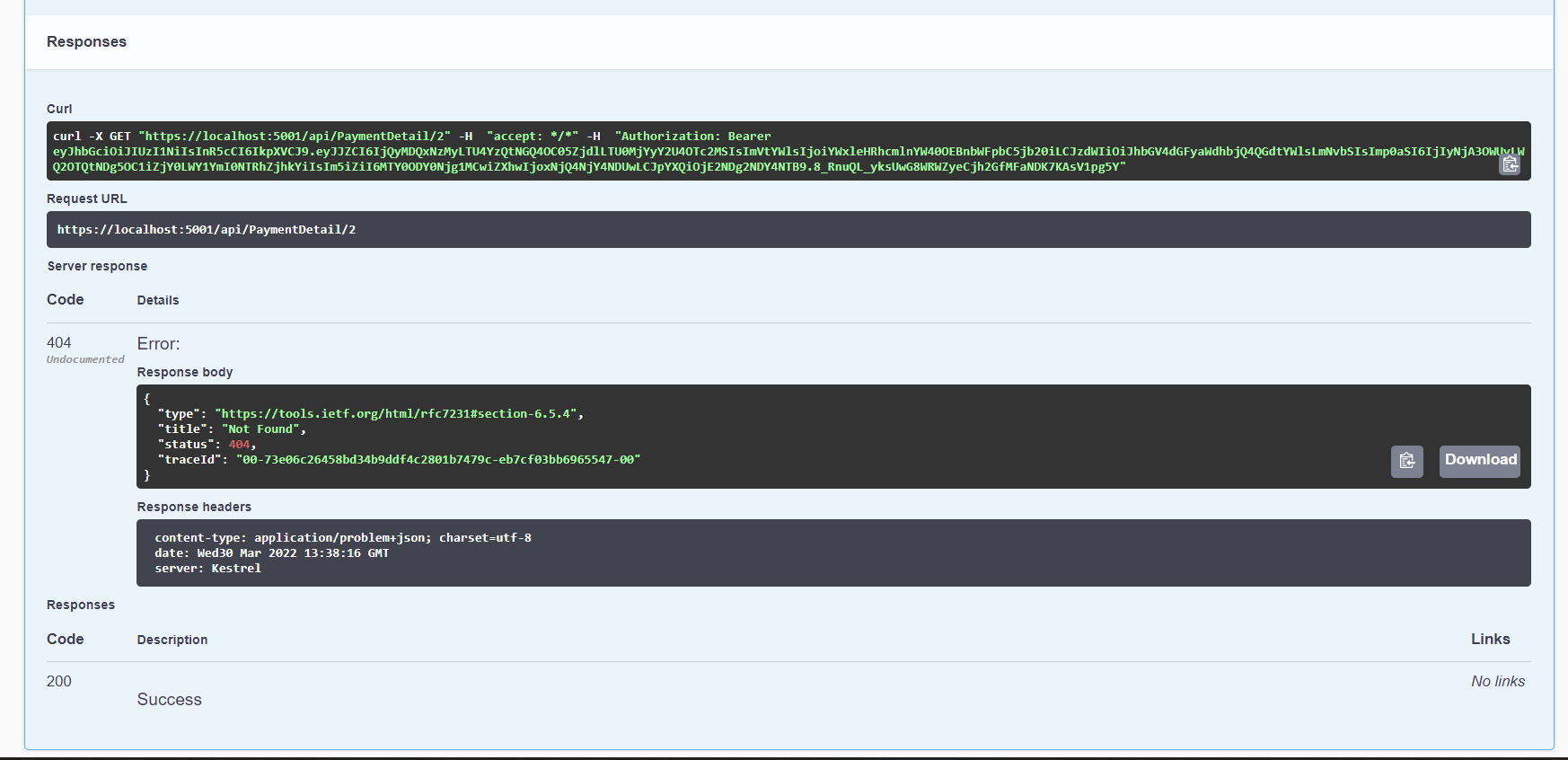


Swagger memberikan response 200(ok)



Cek data yang telah didelete tadi menggunakan get by id

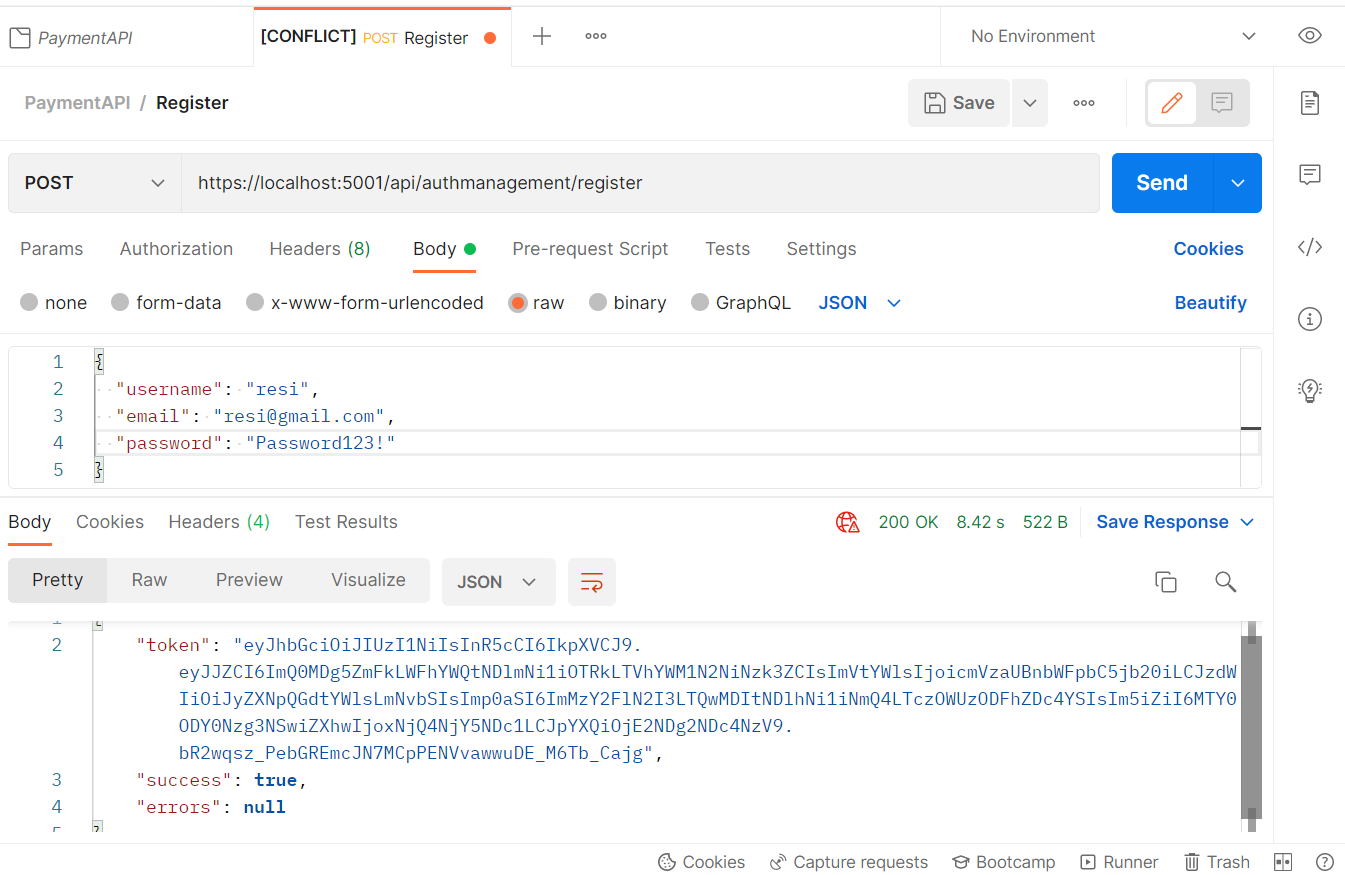
Data berhasil di delete



2. Postman

1. POST Register

-Gunakan fungsi **POST**, masukkan url <https://localhost:5001/api/authmanagement/register>, klik body , lalu pilih **raw**, dan pilih type **JSON**, masukkan username, email, password yang ingin didaftarkan, lalu klik send.



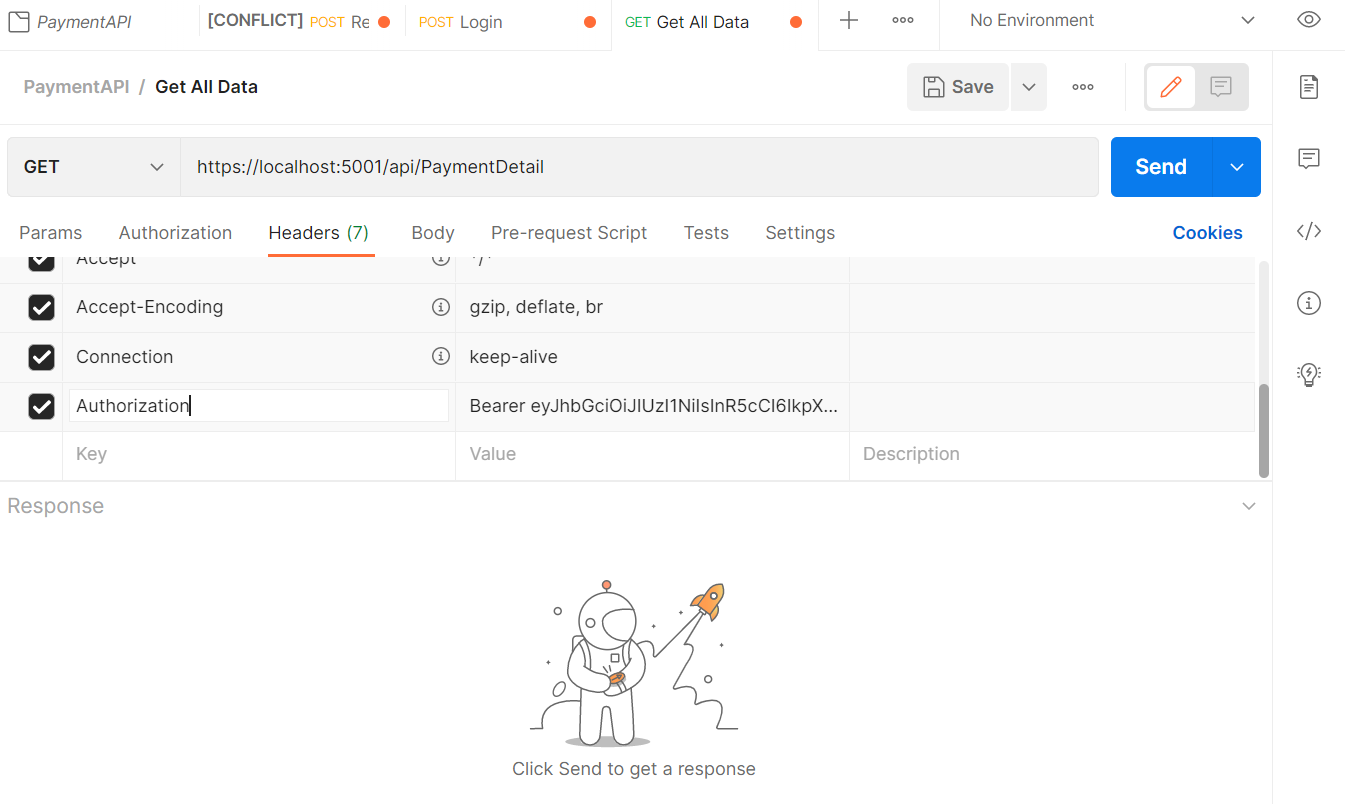
2. POST Login

-Gunakan fungsi **POST**, masukkan url <https://localhost:5001/api/authmanagement/login> , klik body , lalu pilih **raw**, dan pilih type **JSON**, masukkan email, password dengan benar, lalu klik send. Maka postman memberikan response berupa token dan success (200)

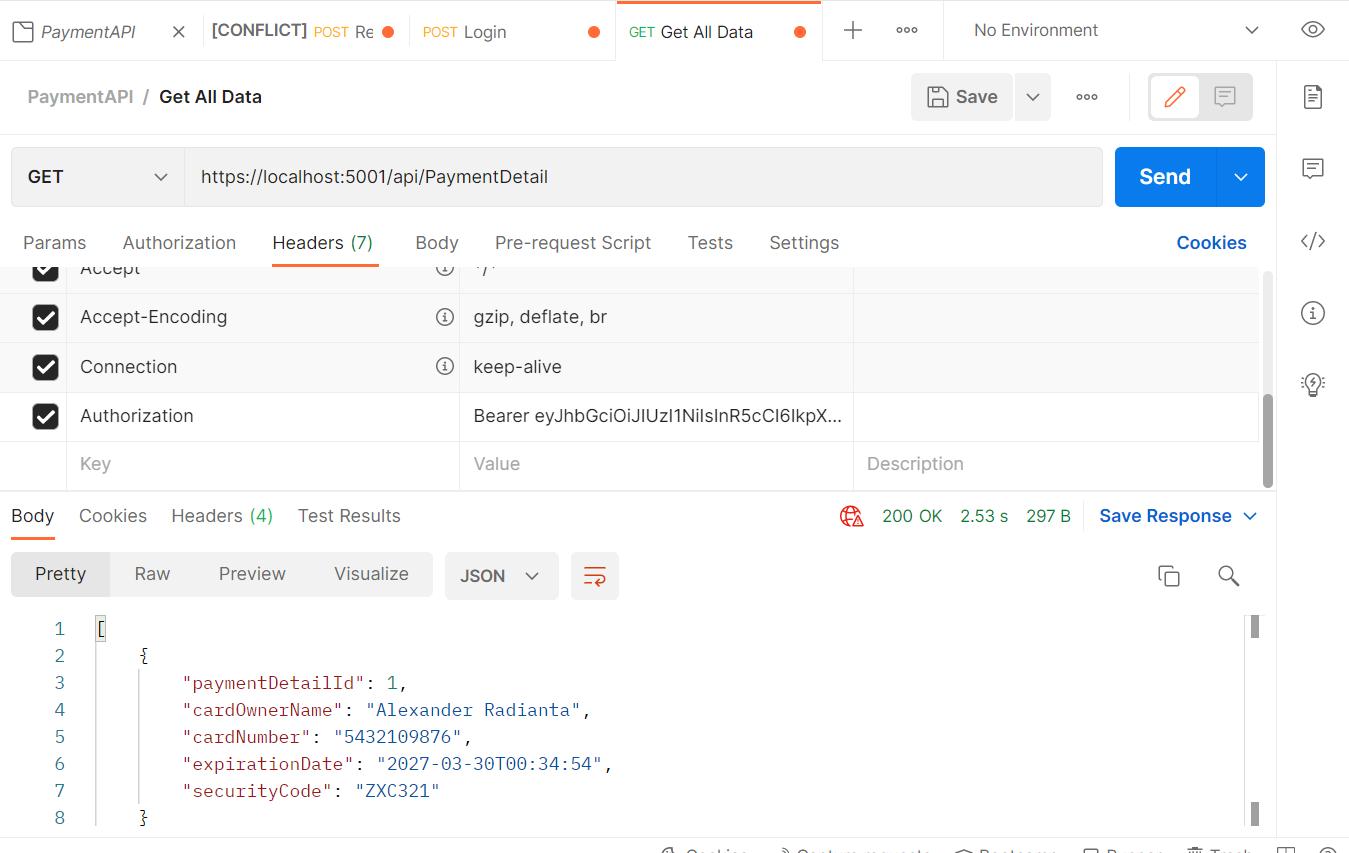


3. Get All Data

-Gunakan fungsi **GET**, masukkan url <https://localhost:5001/api/PaymentDetail>, klik body , lalu pilih **raw**, dan pilih type **JSON**, Simpan token yang didapatkan dari response login tadi, lalu klik headers, tambahkan key baru “Authorization” dengan value “Bearer {token yang didapatkan tadi}, lalu klik tombol send.

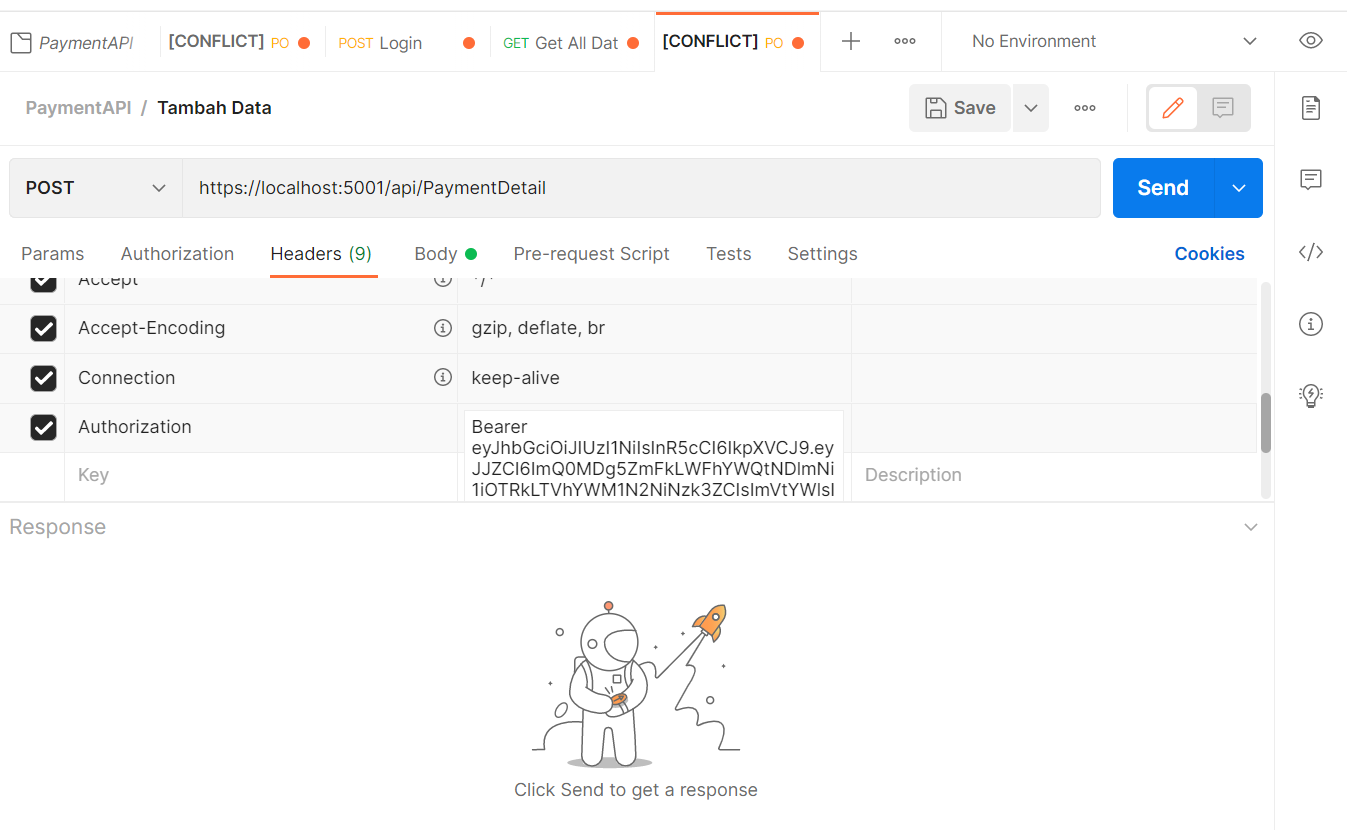


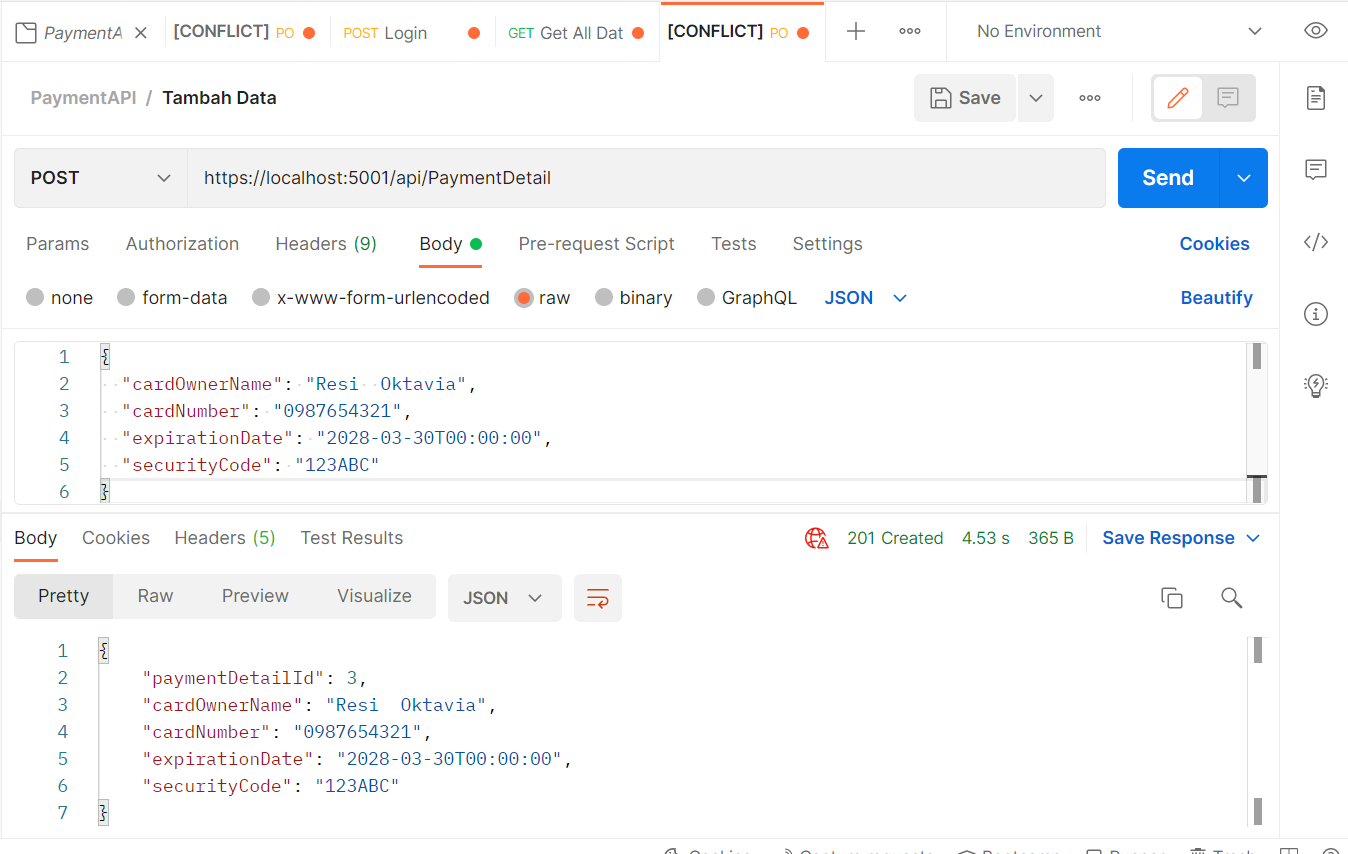
Maka Postman memberikan response berupa data pada database payments.



4. POST Tambah data

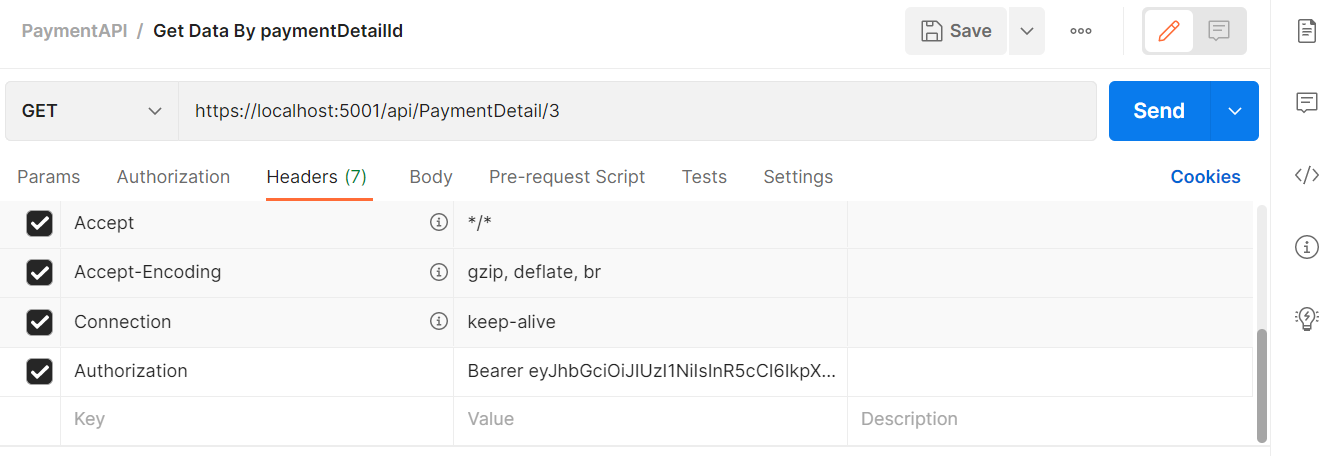
-Gunakan fungsi **POST**, masukkan url [https://localhost:5001/api/PaymentDetail](https://localhost:5001/api/authmanagement/register), lalu klik headers, tambahkan key baru “Authorization” dengan value “Bearer {token yang didapatkan tadi}.



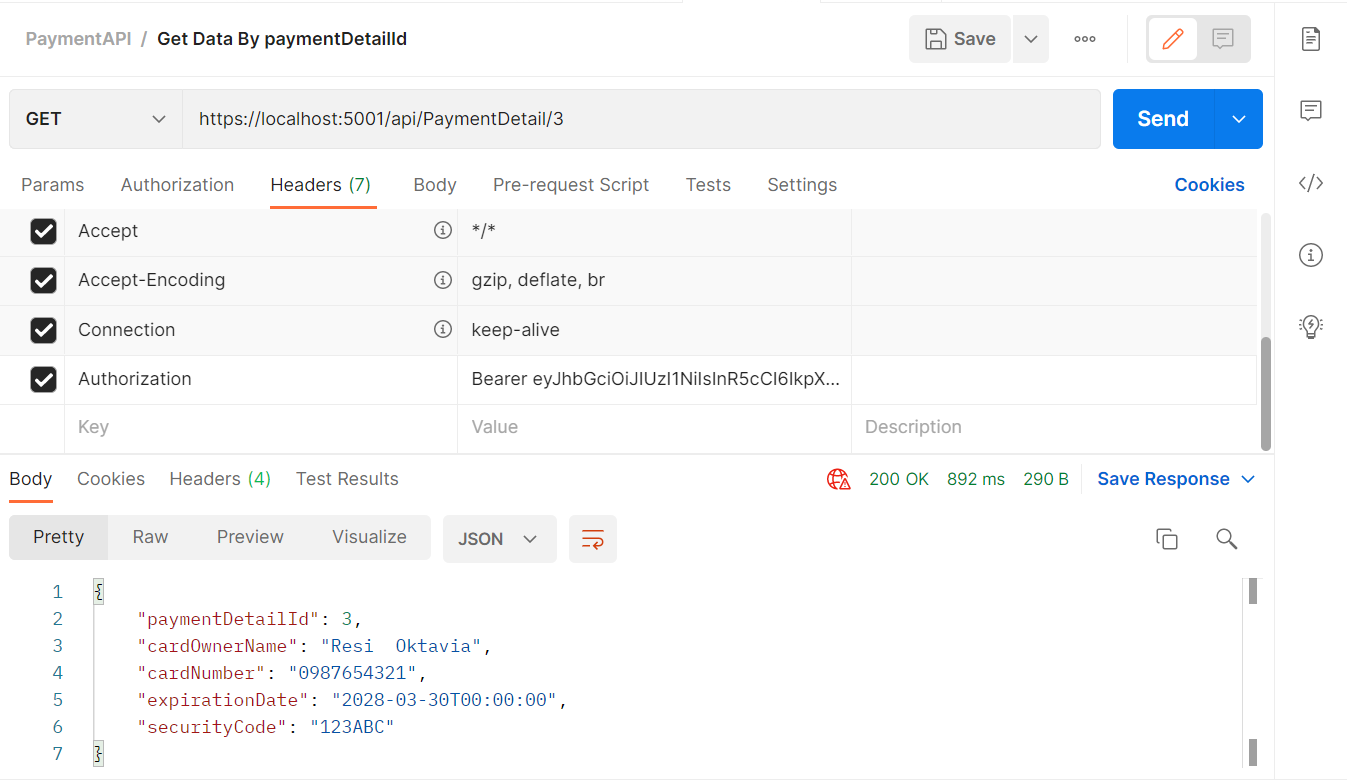
klik body , lalu pilih **raw**, dan pilih type **JSON**, Masukkan data yang ingin dimasukkan sesuai kolom yang ada pada table. Lalu klik tombol send, dan postman memberikan response 201 created, menandakan data berhasil ditambahkan.

5. Get Data By Id

-Gunakan fungsi **GET**, masukkan url <https://localhost:5001/api/PaymentDetail/3> , klik headers, tambahkan key baru “Authorization” dengan value “Bearer {token yang didapatkan tadi}.



Klik tombol send, lalu postman memberikan response 200 dan menampilkan data sesuai paymentDetailId yang di masukkan.

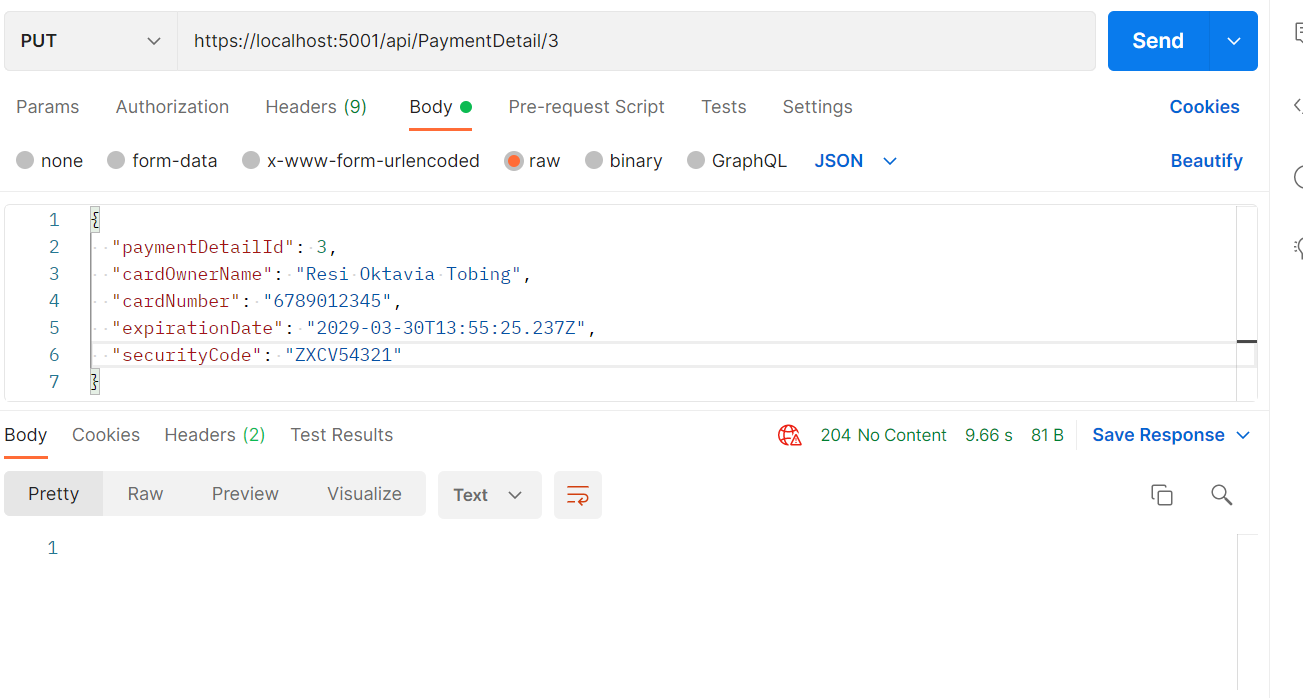


6. PUT(Edit) Data

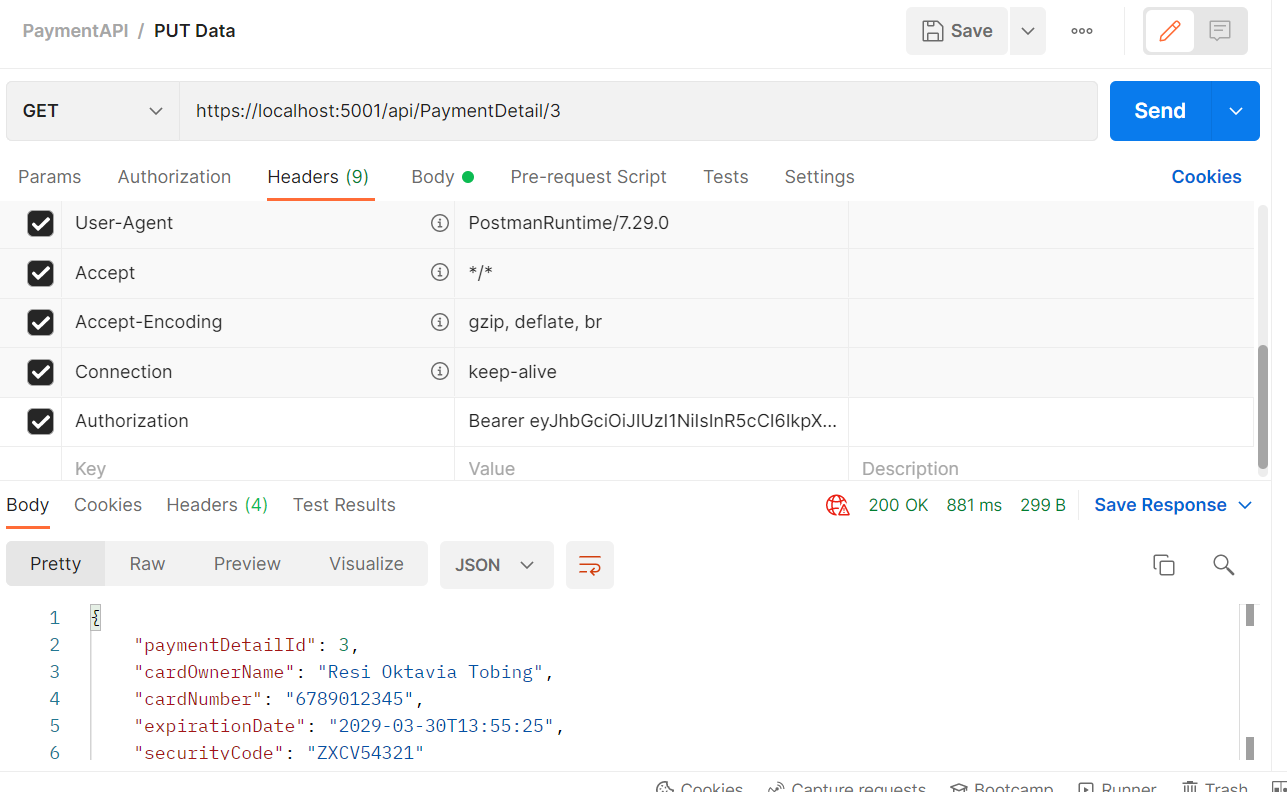
-Gunakan fungsi **PUT**, masukkan url <https://localhost:5001/api/PaymentDetail/3> , lalu klik headers, tambahkan key baru “Authorization” dengan value “Bearer {token yang didapatkan tadi}.



Klik tombol send, lalu postman memberikan response 204 yang menandakan data berhasil diubah

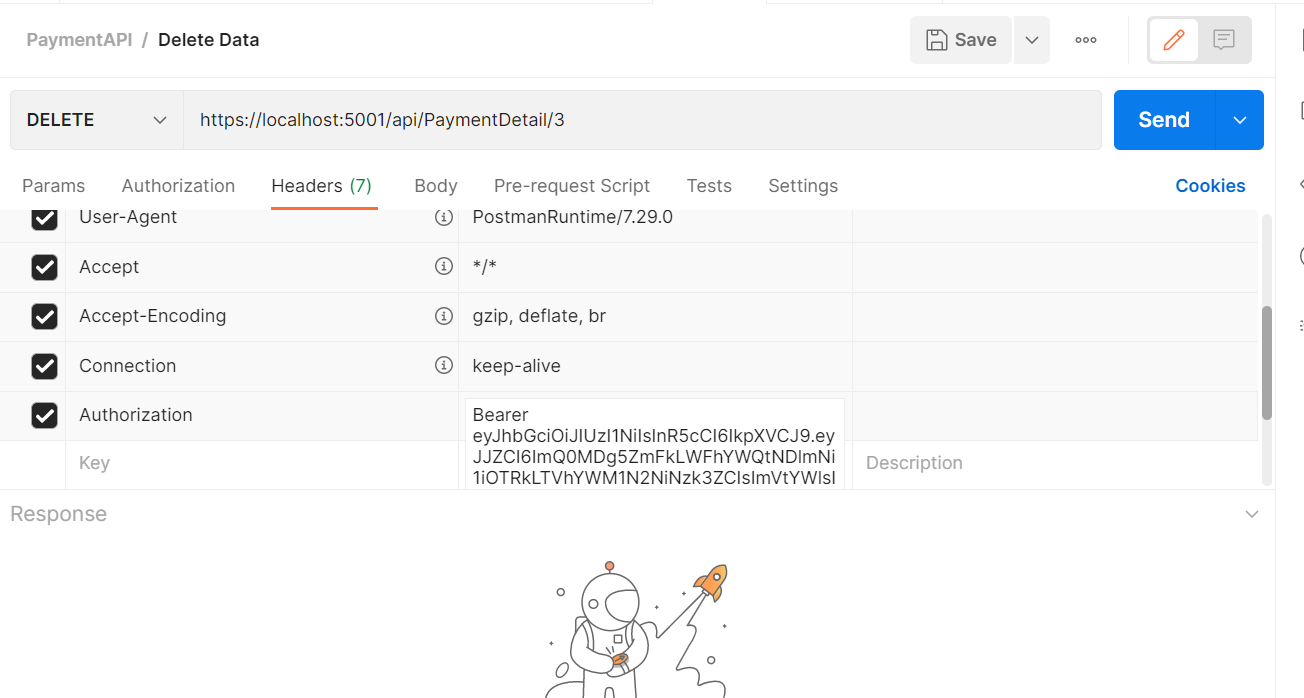


Lakukan pengecekan menggunakan Fungis GET Data By Id

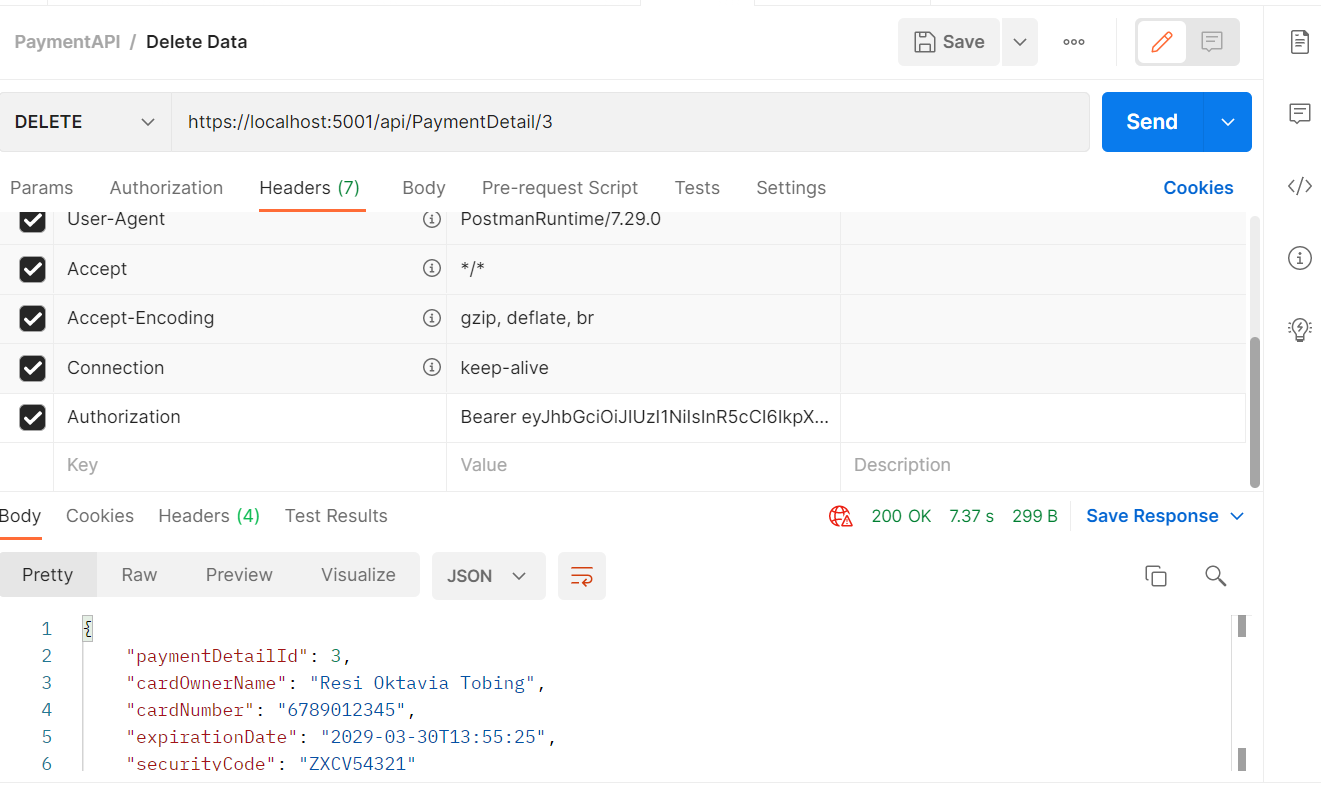


7. Delete Data

-Gunakan fungsi **DELETE**, masukkan url <https://localhost:5001/api/PaymentDetail/3> , lalu klik headers, tambahkan key baru “Authorization” dengan value “Bearer {token yang didapatkan tadi}. Klik tombol Send



Postman memberikan response 200 yang menandakan data berhasil dihapus.



Lakukan pengecekan menggunakan Fungsi GET Data By Id, data not found, karena telah berhasil dihapus.

