Lab 6 : API Security ด้วย JWT โดยใช้บริการของ Auth0

Usa Sammapun, Kasetsart University

ในแลปนี้ เราจะเพิ่มความปลอดภัยให้กับ API โดยทุก request จะต้องส่ง JWT token มาพร้อมกับ request จึงจะสามารถใช้ API ได้ เราสามารถเพิ่มความปลอดภัยด้วย JWT ได้หลายแบบ เช่น

- JWT per user: ต้องส่ง username และ password มาให้ server จากนั้น server จะสร้าง JWT token ให้แต่ละ user ซึ่งจะทำให้ API ปลอดภัยมาก
- JWT per app: ส่ง id และ secret ของแอพมาให้ server จากนั้น server จะสร้าง JWT token ให้แอพนั้น ๆ ซึ่งปลอดภัยพอสมควร แต่ยังน้อยกว่าแบบ JWT per user

เมื่อต้องการใช้ API ผู้ใช้ต้องส่ง JWT token นี้มาพร้อมกับ API request มาที่ server จากนั้น server จะตรวจสอบ JWT token ว่าตรงกับ user หรือ app หรือไม่ ถ้า token ถูกต้องตรงกัน จึงจะเข้าใช้ API function ต่างๆ ได้

การสร้าง JWT token สามารถทำได้ 2 ช่องทาง คือ

- server ทำเอง
- ใช้บริการ Auth Service บน cloud เช่น Auth0, Amazon Cognito

ในแลปนี้ เราจะใช้ JWT per app และใช้บริการ Auth0 บน cloud ในการสร้าง JWT ให้เรา

I. เพิ่ม Spring Security ใน pom.xml

1. เพิ่ม dependency ดังนี้
    - Spring Security Starter และ Spring Security OAuth
    - Security Testing
2. Reload pom.xml หลังเพิ่ม dependency ด้วย

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-oauth2-resource-server</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-oauth2-jose</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-test</artifactId>
  <scope>test</scope>
</dependency>
```

II. เพิ่ม security configuration

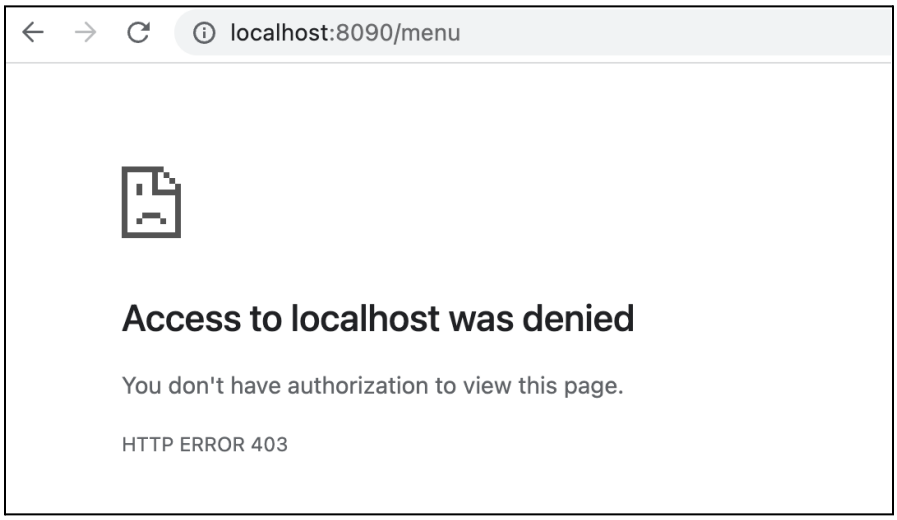
1. สร้าง package security
2. เพิ่มคลาส SecurityConfig ใน package นี้
    - CSRF เป็นความไม่ปลอดภัยที่เกิดขึ้นกับ frontend application เท่านั้น สำหรับ API จึง disable
    - โค้ด `http.authorizeRequests()` หมายความว่า เราต้องการควบคุมการเข้าถึงโปรแกรมนี้ โดย request ใด ๆ ต้องมีการ authenticate ก่อน

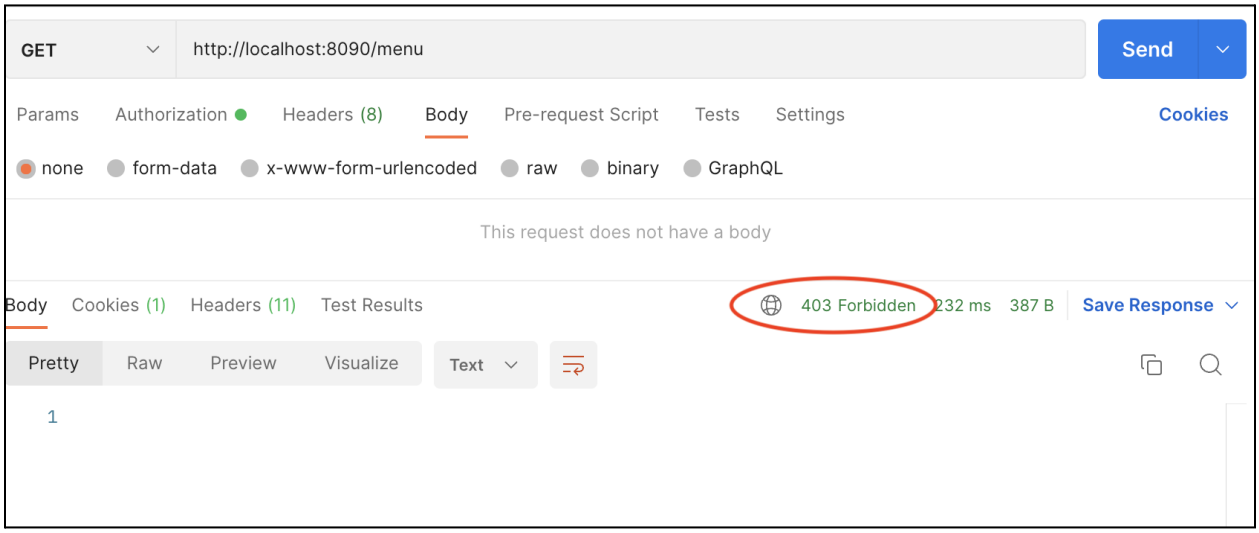
```
package th.ac.ku.menu.security;

import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigur
erAdapter;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

   @Override
   protected void configure(HttpSecurity http) throws Exception {
       http
               .csrf().disable()
               .authorizeRequests()
               .anyRequest()
               .authenticated();
   }
}
```

3. Run โปรแกรม (คลิกรันที่ MenuApplication)
    - ถ้าไม่ได้ใช้ IntelliJ และรันผ่าน command line ให้ใช้คำสั่งต่อไปนี้
    - (ต้องดาวน์โหลด Apache Maven ก่อน https://maven.apache.org/download.cgi)

```
mvn spring-boot:run
```

4. ลอง GET หรือ POST
    - จะได้ error code 403 Forbidden



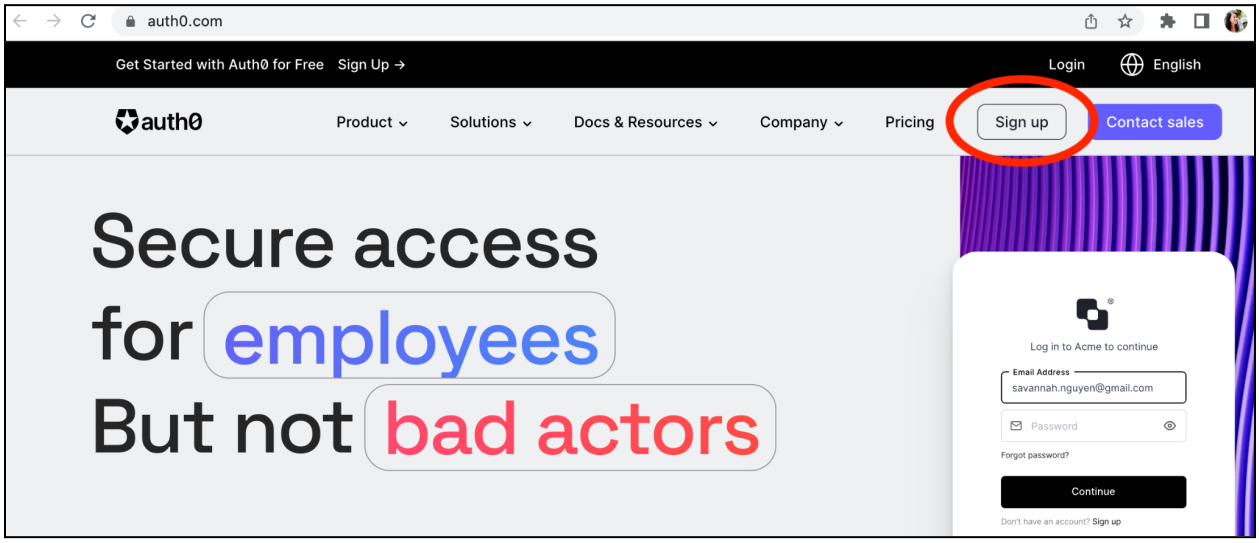


III. Signup with Auth0

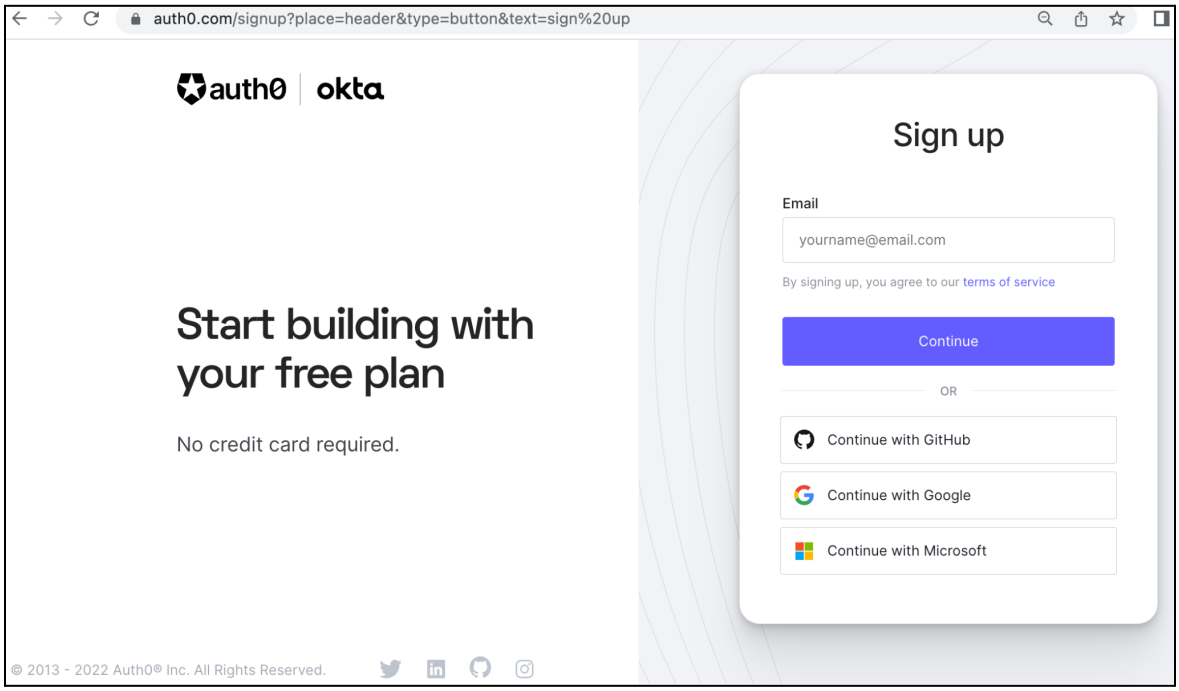
Ref: https://auth0.com/docs/quickstart/backend/java-spring-security5/01-authorization

ขั้นตอนถัดไป จะเป็นการลงทะเบียนและเข้าใช้บริการ Auth0

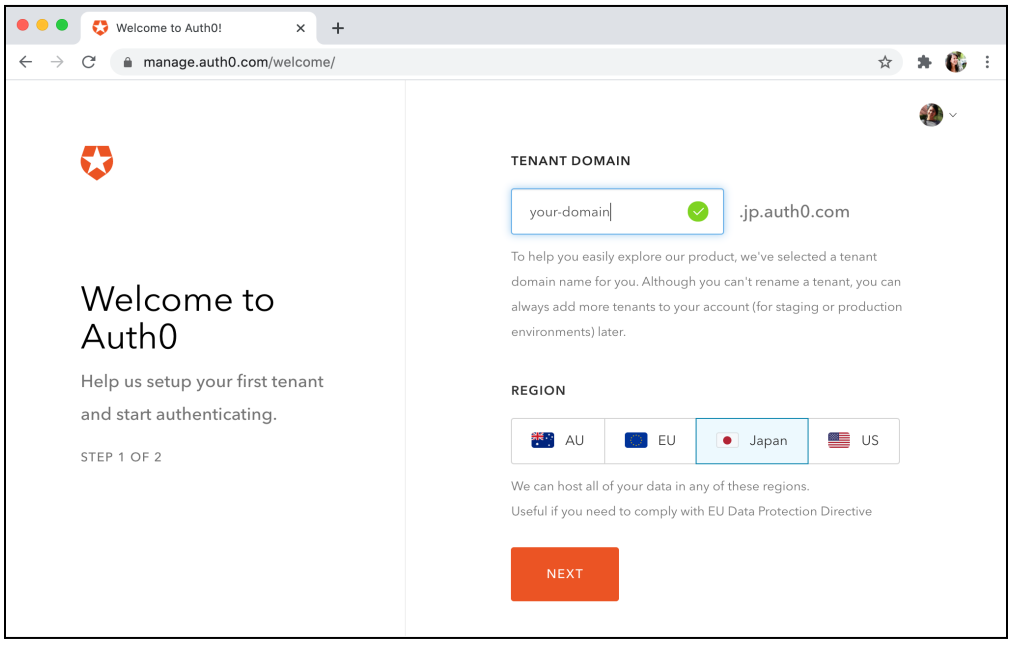
1. ไปที่ Auth0 URL นี้ และกดปุ่ม Sign up: https://auth0.com/



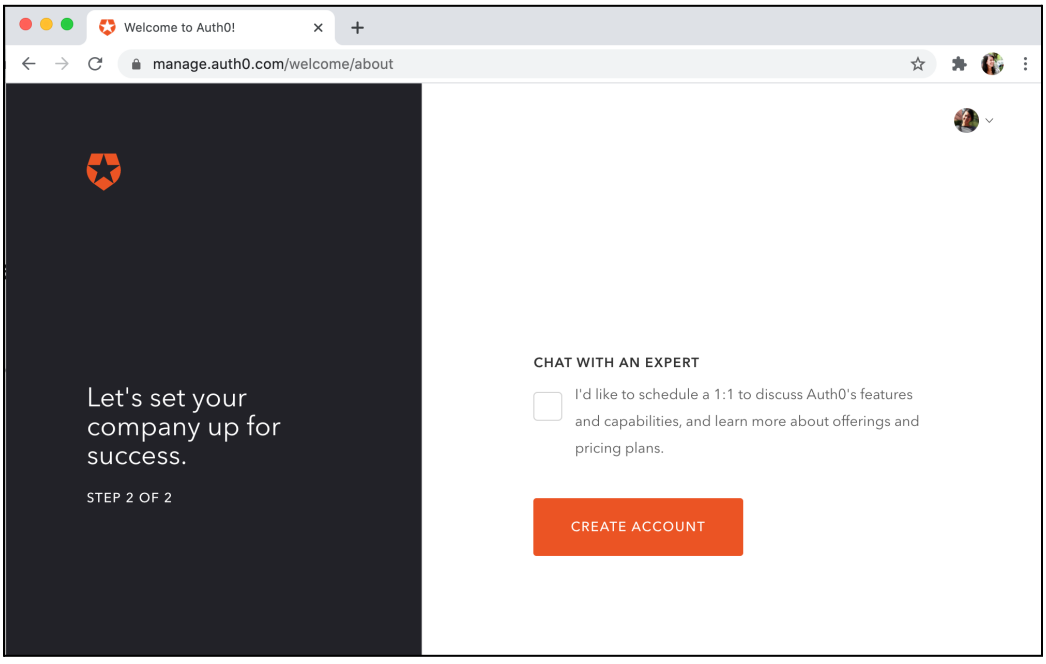
2. Signup ด้วยวิธีใดก็ได้



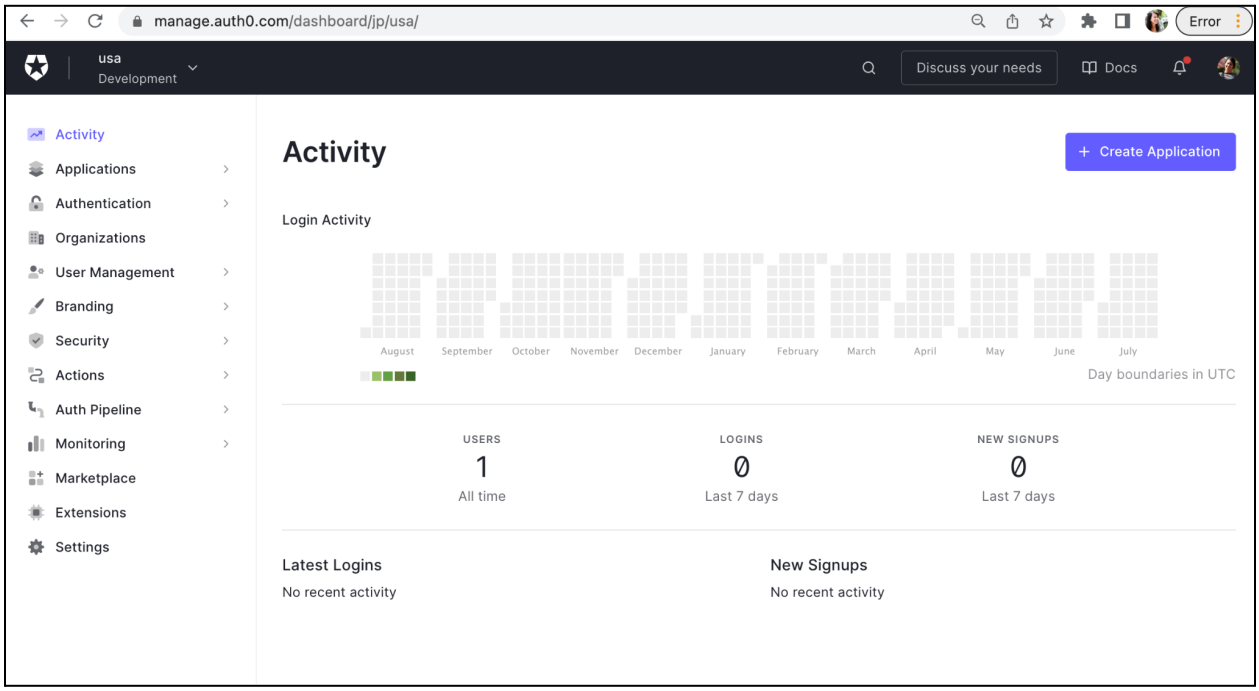
3. เลือก tenant domain name ตามต้องการ (คล้ายกับ domain name ใน Jira) และคลิก NEXT
    - “A tenant is a container that Auth0 uses to store your identity service config and your users in isolation — no other Auth0 customer can peek into or access your tenant. It's similar to you being a tenant in an apartment building.”



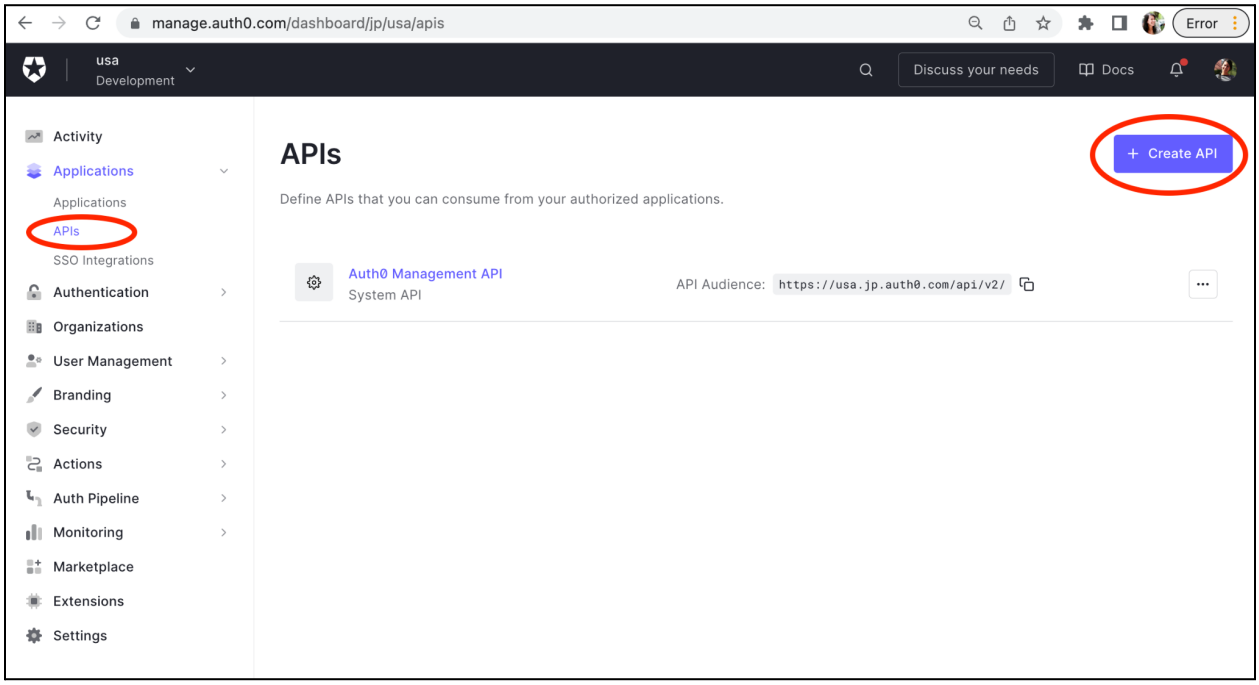
4. กดปุ่ม CREATE ACCOUNT



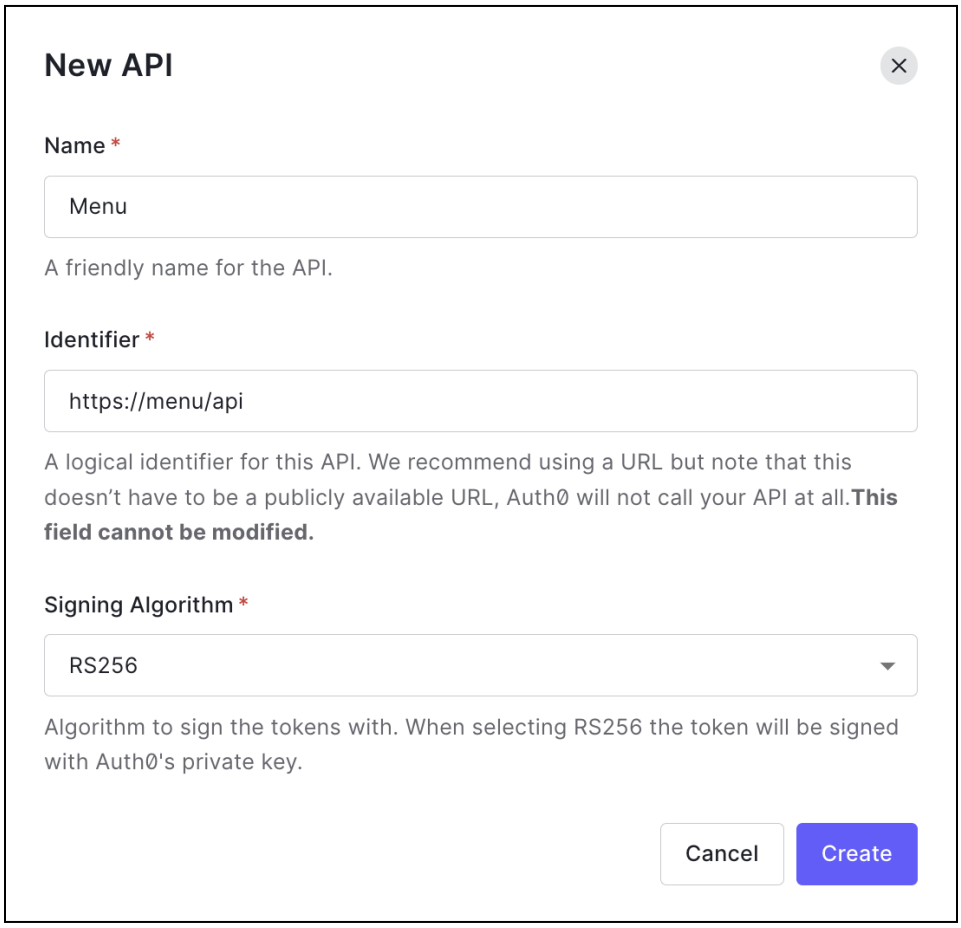
5. เมื่อลงทะเบียนเสร็จแล้ว จะได้หน้า Dashboard



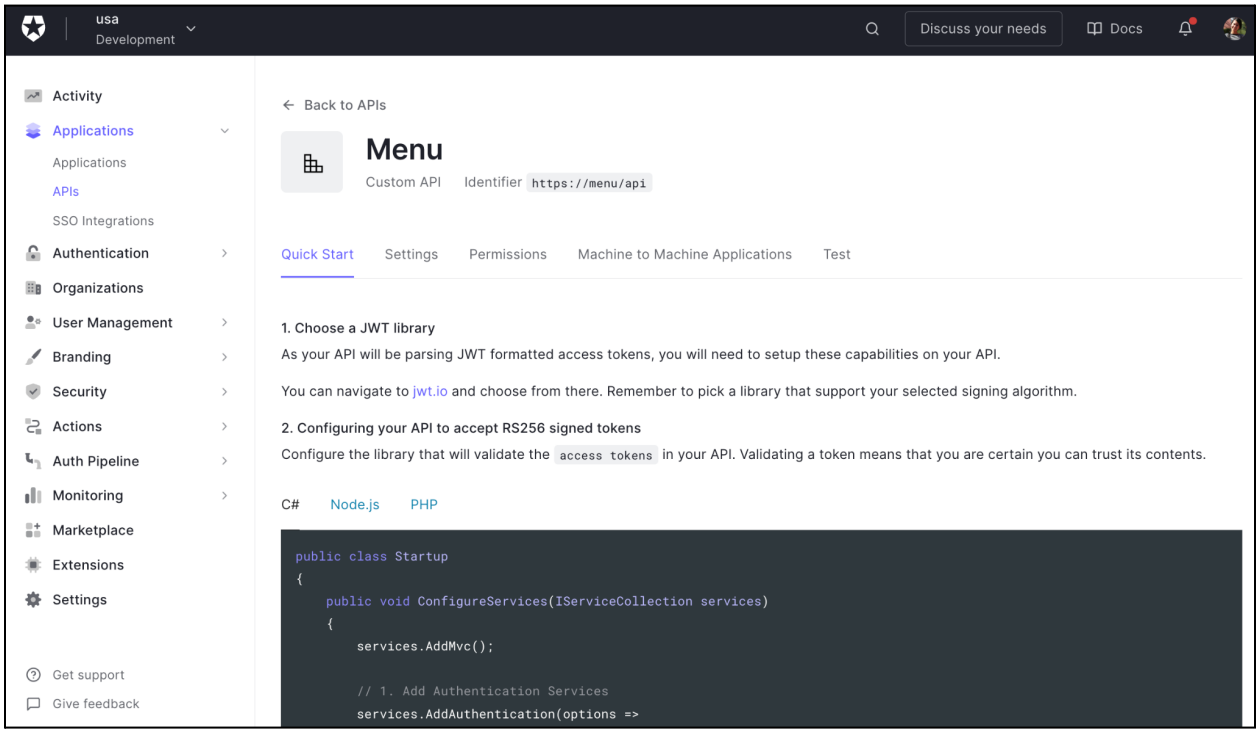
6. ที่เมนูซ้ายมือ ให้กดเมนู Applications → APIs
7. จากนั้นให้กดปุ่ม “+Create API” ขวาบน



8. กรอกข้อมูล API ของเรา ดังรูป
    - Name: เป็นชื่อ API
    - Identifier: ใช้ในการบอก Auth0 ว่าจะใช้ API ใด เป็นเหมือน id ของ API
    - Signing Algorithm: เป็น algorithm ในการเข้ารหัสข้อมูลแบบหนึ่ง เลือก RS256 ซึ่งปลอดภัย



9. หลังกดปุ่ม Create จะได้หน้าแสดงรายละเอียดของ API

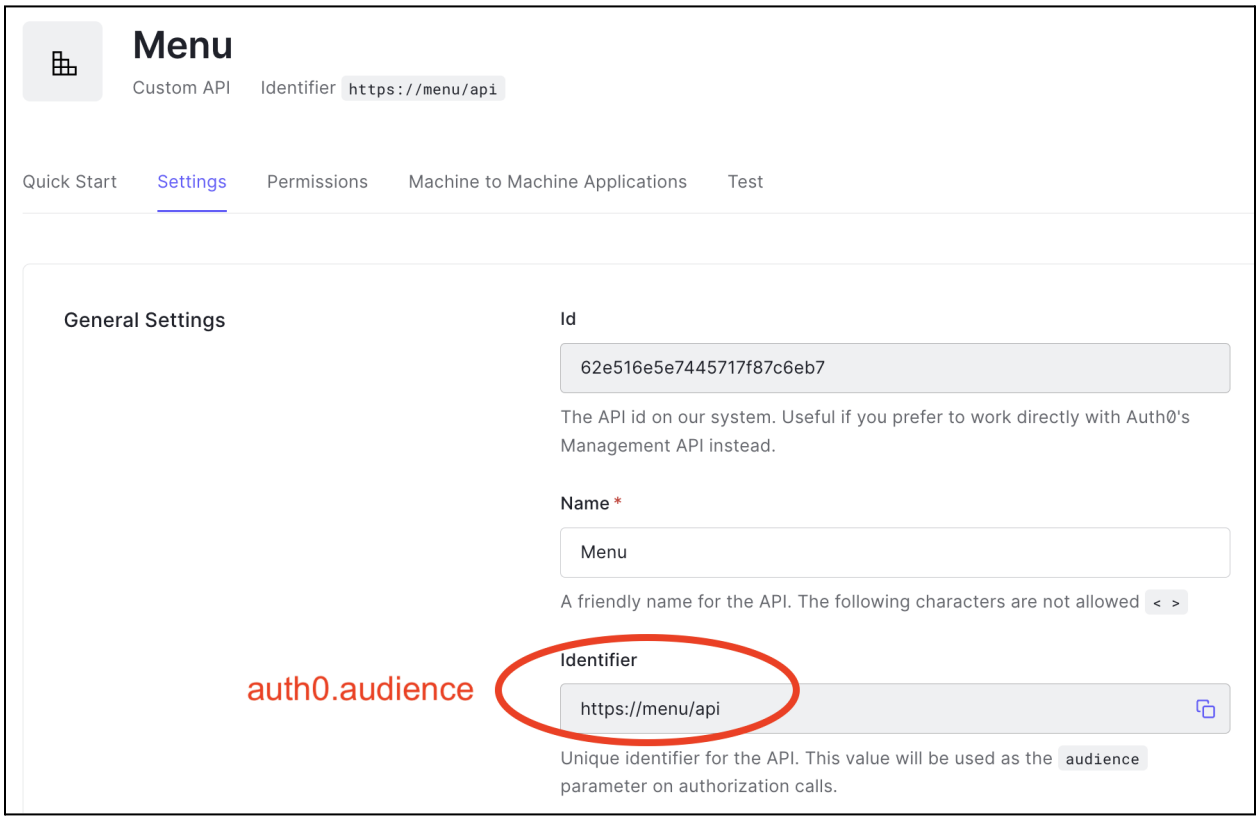


IV. เชื่อม API ของเรากับ Auth0

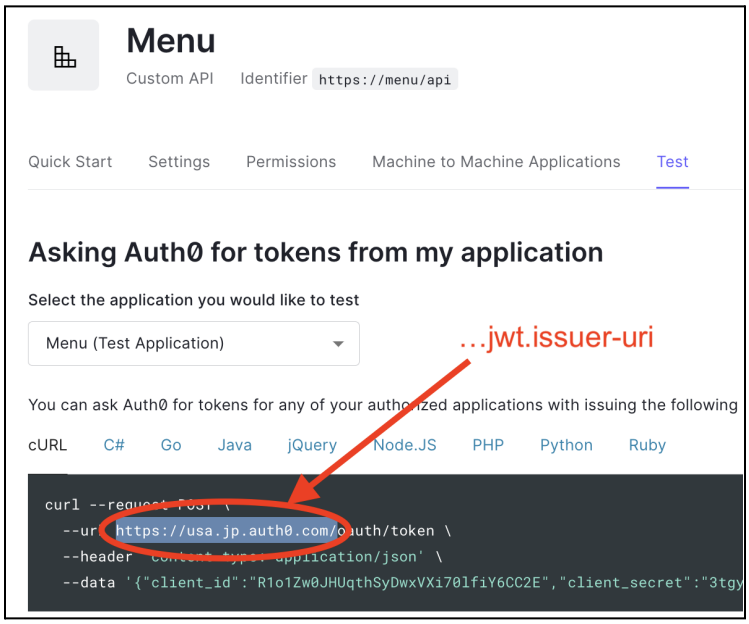
1. เพื่อ config เกี่ยวกับ Auth0 ที่ **application.properties** (เติมไปท้ายไฟล์ได้)
    - ดูข้อ 2 เพื่อกรอกข้อมูล audience และ JWT issuer

```
# Auth0
auth0.audience=
spring.security.oauth2.resourceserver.jwt.issuer-uri=
```

2. สามารถหาข้อมูล audience and JWT issuer โดยกดที่แท็บ Settings ที่หน้ารายละเอียด API
    a. โดยข้อมูล **auth0.audience** อยู่ที่ **Identifier** field



b. กดที่แท็บ **Test** โดยข้อมูล **jwt.issuer-uri** จะอยู่ตามรูปด้านล่าง
   i. ต้องมีเครื่องหมาย / ต่อท้าย URL ด้วย
   ii. เช่น https://usa.jp.auth0.com/



3. สร้างคลาส AudienceValidator ใน security package
   a. คลาสนี้จะตรวจสอบว่า JWT token ที่ API ได้รับมานั้น ถูกต้องไปหรือไม่ (คือ มีข้อมูล audience ที่ตรงกับที่เรา config ไว้ที่ application.properties หรือไม่)

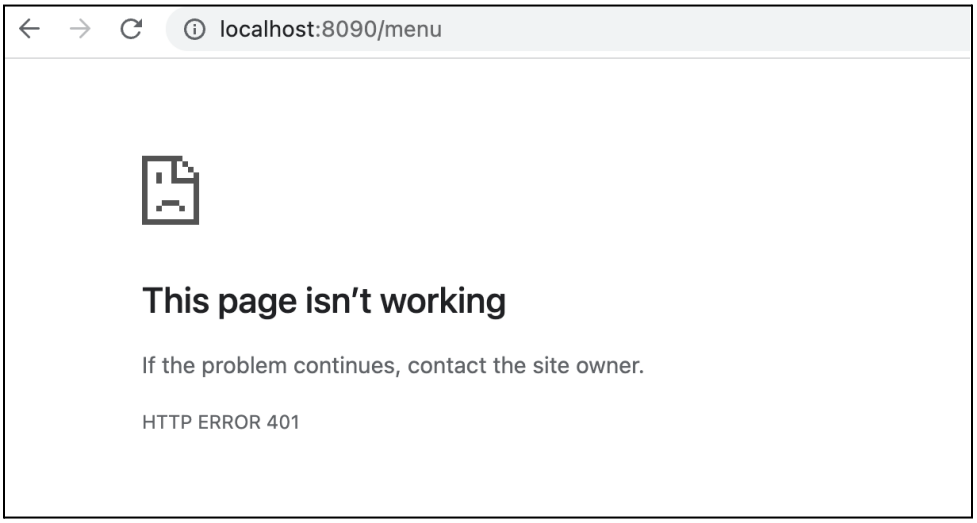
```
package th.ac.ku.menu.security;

import org.springframework.security.oauth2.core.OAuth2Error;
import org.springframework.security.oauth2.core.OAuth2ErrorCodes;
import org.springframework.security.oauth2.core.OAuth2TokenValidator;
import org.springframework.security.oauth2.core.OAuth2TokenValidatorResult;
import org.springframework.security.oauth2.jwt.Jwt;
import java.util.List;

public class AudienceValidator implements OAuth2TokenValidator<Jwt> {
   private final String audience;

   AudienceValidator(String audience) {
       this.audience = audience;
   }

   public OAuth2TokenValidatorResult validate(Jwt jwt) {
       List<String> audiences = jwt.getAudience();

       if (audiences.contains(this.audience)) {
           return OAuth2TokenValidatorResult.success();
       }
       OAuth2Error err = new OAuth2Error(OAuth2ErrorCodes.INVALID_TOKEN);
       return OAuth2TokenValidatorResult.failure(err);
   }
}
```

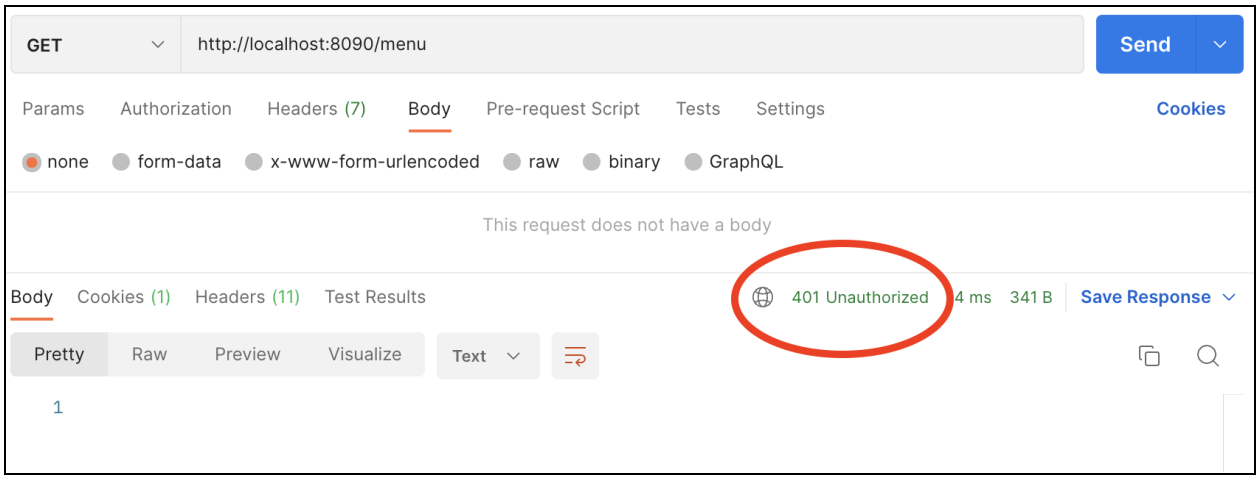
4. เปลี่ยน security config ให้ตรวจสอบ JWT token สำหรับการทำ authentication
    a. และเพิ่มเมธอด Jwt decoder เพื่อสร้างอ๊อบเจ็ค JwtDecoder ในการถอดรหัสจาก JWT token ที่ได้รับมา
    b. **หมายเหตุ:** ให้ import `@Value` จาก `org.springframework.beans.factory.annotation.Value;`

```
package th.ac.ku.menu.security;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecu
rity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityCo
nfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import
org.springframework.security.oauth2.core.DelegatingOAuth2TokenValidator;
import org.springframework.security.oauth2.core.OAuth2TokenValidator;
import org.springframework.security.oauth2.jwt.*;


@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Value("${auth0.audience}")
    private String audience;

    @Value("${spring.security.oauth2.resourceserver.jwt.issuer-uri}")
    private String issuer;

    @Override
    protected void configure(HttpSecurity http) throws Exception {

        http
                .csrf().disable()
                .authorizeRequests()
                .anyRequest()
                .authenticated() // เอา ; ออกด้วยค่ะ

        // use stateless session, so user's state is not stored
        .and()
                .sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)

        // use oauth as a resource server to do jwt validation
        .and()
                .oauth2ResourceServer()
                .jwt()
                .decoder(jwtDecoder());
    }
```

```
    private JwtDecoder jwtDecoder() {
        OAuth2TokenValidator<Jwt> withAudience =
                                        new AudienceValidator(audience);

        OAuth2TokenValidator<Jwt> withIssuer =
              JwtValidators.createDefaultWithIssuer(issuer);

        OAuth2TokenValidator<Jwt> validator =
              new DelegatingOAuth2TokenValidator<>(withAudience, withIssuer);

        NimbusJwtDecoder jwtDecoder = (NimbusJwtDecoder)
                                    JwtDecoders.fromOidcIssuerLocation(issuer);

        jwtDecoder.setJwtValidator(validator);

        return jwtDecoder;
    }
}
```

5. ลอง GET หรือ POST จะได้ Error เป็น 401 unauthorized

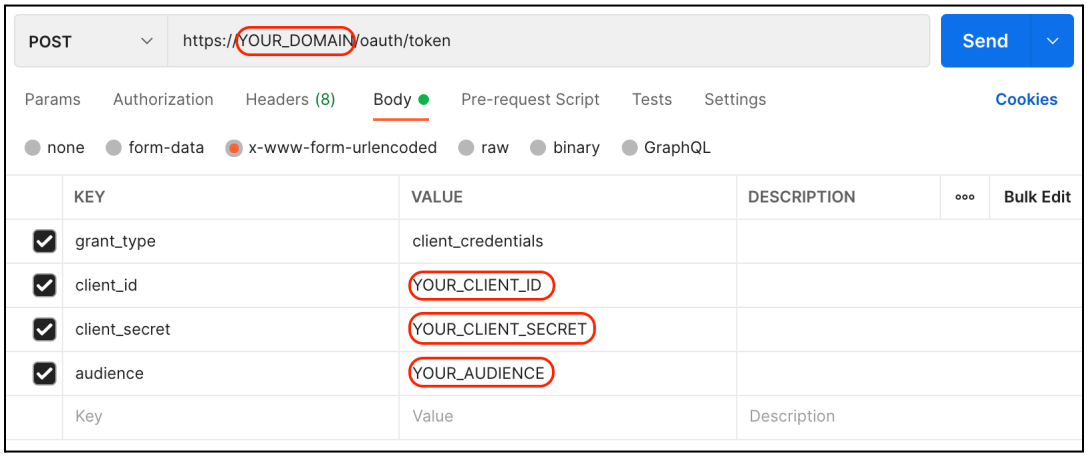




V. รับ Token ที่สร้างจาก Auth0[1]

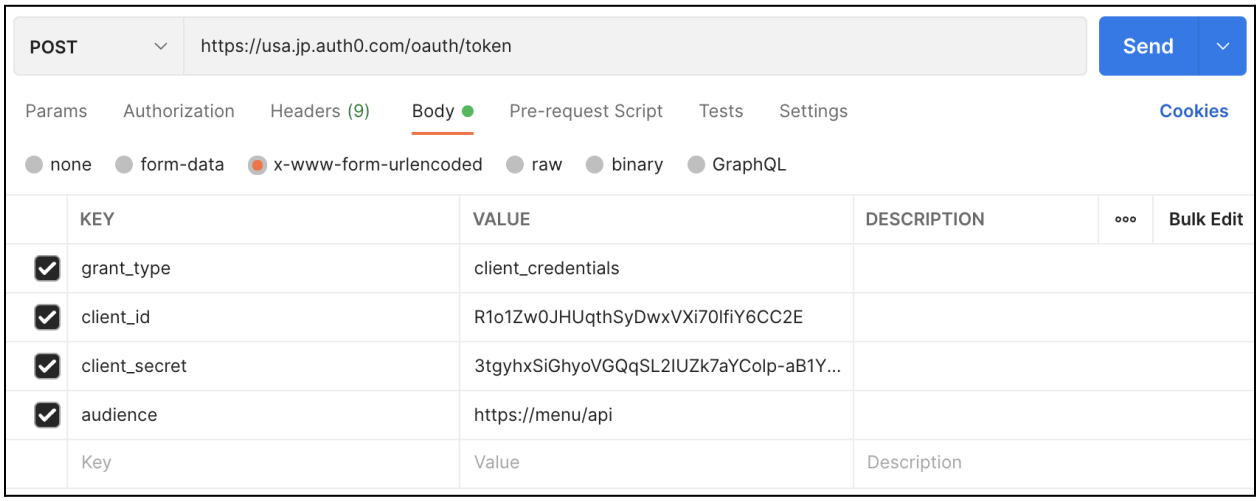
1. เปิดโปรแกรม Postman
2. ให้ POST ไปที่ URL นี้
    - https://YOUR_DOMAIN/oauth/token
        i. เช่น https://usa.jp.auth0.com/oauth/token
    - ไปที่แท็บ Body ให้เลือก “x-www-form-urlencoded” และใส่ข้อมูล key-value ต่อไปนี้
        i. (ข้อมูล client_id, client_secret อยู่ในข้อ c. โดยข้อมูล audience จะเหมือนกับข้อมูล auth0.audience ใน application.properties)

grant_type:	client_credentials
client_id:	YOUR_CLIENT_ID
client_secret:	YOUR_CLIENT_SECRET
audience:	YOUR_AUDIENCE

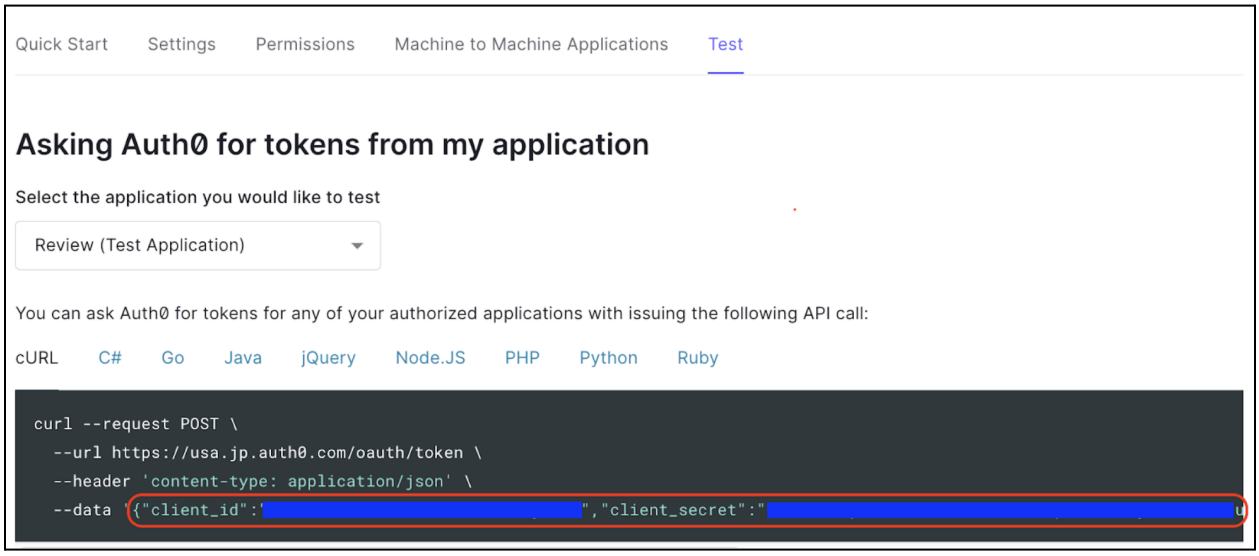


ตัวอย่างของอาจารย์ตามรูปต่อไปนี้ ให้นิสิตหาข้อมูลของตนเอง

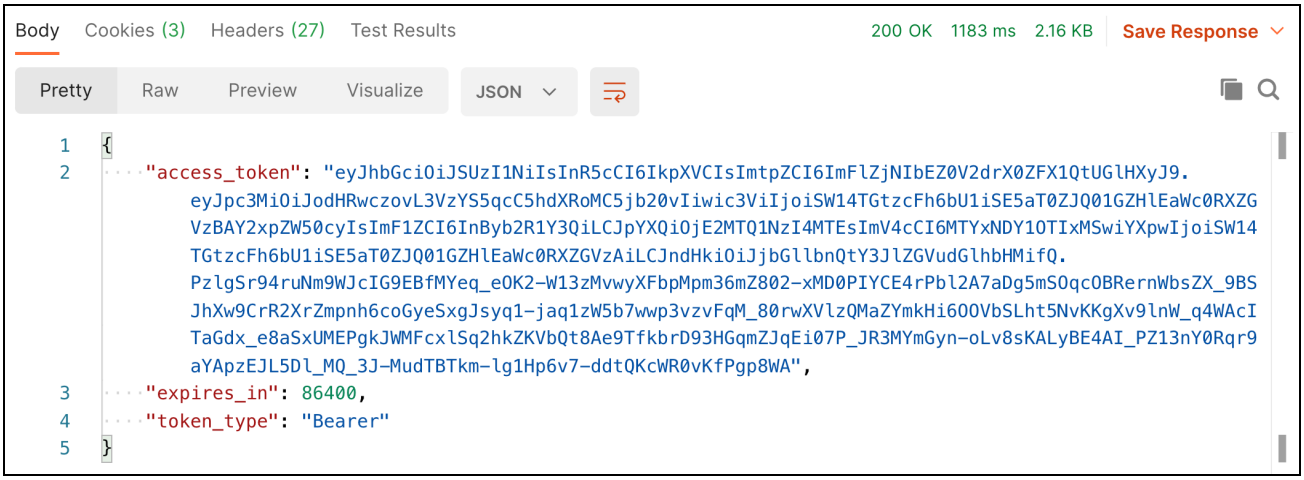
[1] Ref: https://auth0.com/docs/quickstart/backend/java-spring-security5/02-using



- ไปที่หน้า API ที่เว็บ Auth0 จากนั้นไปที่แท็บ Test
  i. จะเห็นข้อมูล **client id** และ **client secret**
  ii. ให้กรอกข้อมูลที่ Postman และกด Send

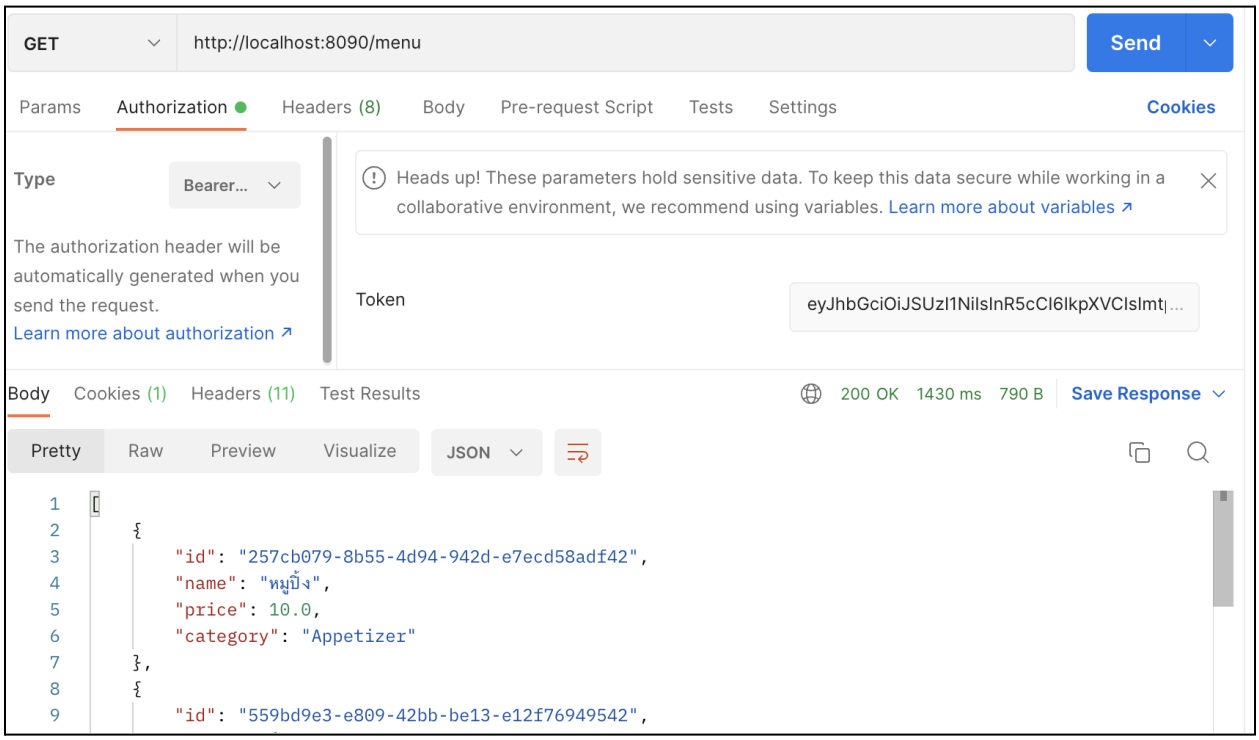


3. หลังจากกด Send นิสิตจะได้ JWT Token คล้ายกับรูปด้านล่าง
   - ให้ Copy ข้อมูลใน access token field.


Body
Cookies (3)
Headers (27)
Test Results
200 OK
1183 ms
2.16 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6ImFlZjNIbEZ0V2drX0ZFX1QtUGlHXyJ9.
eyJpc3MiOiJodHRwczovL3VzYS5qcC5hdXRoMC5jb20vIiwic3ViIjoiSW14TGtzcFh6bU1iSE5aT0ZJQ01GZHlEaWc0RXZG
VzBAY2xpZW50cyIsImF1ZCI6InByb2R1Y3QiLCJpYXQiOjE2MTQ1NzI4MTEsImV4cCI6MTYxNDY1OTIxMSwiYXpwIjoiSW14
TGtzcFh6bU1iSE5aT0ZJQ01GZHlEaWc0RXZGVzAiLCJndHkiOiJjbGllbnQtY3JlZGVudGlhbHMifQ.
PzlgSr94ruNm9WJcIG9EBfMYeq_eOK2-W13zMvwyXFbpMpm36mZ802-xMD0PIYCE4rPbl2A7aDg5mSOqcOBRernWbsZX_9BS
JhXw9CrR2XrZmpnh6coGyeSxgJsyq1-jaq1zW5b7wwp3vzvFqM_80rwXVlzQMaZYmkHi6OOVbSLht5NvKKgXv9lnW_q4WAcI
TaGdx_e8aSxUMEPgkJWMFcxlSq2hkZKVbQt8Ae9TfkbrD93HGqmZJqEi07P_JR3MYmGyn-oLv8sKALyBE4AI_PZ13nY0Rqr9
aYApzEJL5Dl_MQ_3J-MudTBTkm-lg1Hp6v7-ddtQKcWR0vKfPgp8WA",
"expires_in": 86400,
"token_type": "Bearer"
}

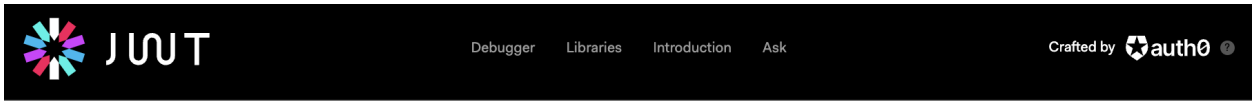
4. ลอง GET หรือ POST
    - ไปที่แท็บ “**Authorization**”
    - เลือก “**Bearer Token**”
    - ให้ paste ข้อมูล access token จากข้อที่แล้วใน field ด้านขวามือ

5. จะเห็นว่า เราสามารถใช้ API ได้แล้ว



6. นิสิตสามารถลองส่ง API แบบไม่มี JWT token เราจะได้ 401 error.

7. ไปที่เว็บ: https://jwt.io/
    - เป็นเว็บที่ใช้ถอดรหัส JWT Token



Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6ImFlZjNIbEZ0V2drX0ZFX1QtUGlHXyJ9.eyJpc3MiOiJodHRwczovL3VzYS5qcC5hdXRoMC5jb20vIiwic3ViIjoiUjFvMVp3MEpIVXF0aFN5RHd4VlhpNzBsZmlZNkNDMkVAY2xpZW50cyIsImF1ZCI6Imh0dHBzOi8vbWVudS9hcGkiLCJpYXQiOjE2NTkxODcxMTMsImV4cCI6MTY1OTI3MzUxMywiYXpwIjoiUjFvMVp3MEpIVXF0aFN5RHd4VlhpNzBsZmlZNkNDMkUiLCJndHkiOiJjbGllbnQtY3JlZGVudGlhbHMifQ.lDlpjzOVLv5P2lcp7Sqekojj1TUQ7cQU-yIrJazZ_dmhvNRsAkUbG2ceBP3DSDTvvLUHQ5kH_D_RfoVoe5pR6wRq-l-ivlJclSltmrG9bekVWxQ4OhlSQAdhUAa1t4PvL-mTZ3TUabnFiiMG4mzfZZJWlg9YOLvWd6v5JC6HptlAqbOeMbq5MADylg61-73bbThrIxo0U-cconFcXeO-chrlCVjdz8Om5U5CQ8ZE5rPnM52zrb2-Y8nYIEkuCDJXU6o89Jxg8q_mNzpmvuynhXJ8fSlKeHGpXbPVWszjDi5vJWalX97O3j49EF36K9sQ2dAu-hEzeapHj5ZJshUsg
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE


```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "aef3HlFtWgk_FE_T-PiG_"
}
```


PAYLOAD: DATA


```
{
  "iss": "https://usa.jp.auth0.com/",
  "sub": "R1o1Zw0JHUqthSyDwxVXi70lfiY6CC2E@clients",
  "aud": "https://menu/api",
  "iat": 1659187113,
  "exp": 1659273513,
  "azp": "R1o1Zw0JHUqthSyDwxVXi70lfiY6CC2E",
  "gty": "client-credentials"
}
```


VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  {
    "e": "AQAB",
    "kty": "RSA",
    "n": "wJNS93NA53FU2c4Up-Qt4x
```