Lab 10 : Spring Thymeleaf – Validation and Connect to API

Usa Sammapun, Kasetsart University

ในแลปส่วนที่แล้ว เราเรียนรู้การ signup และ login
ในแลปส่วนนี้ เรียนเพิ่มเติมดังนี้

- Lombok – library และ plugin ที่มี annotation แทนการสร้าง getter, setters, ฯลฯ
- DTO (data transfer object) – แยก object ที่ใช้ส่งให้/รับจาก client ออกจาก object ที่จะ save ลง database
- Input validation – Spring Boot มี library ที่ช่วยตรวจสอบ input
- เชื่อมต่อไปที่ menu API ที่เราสร้างเอาไว้ เพื่อนำข้อมูลรายการอาหารมาใช้
- แยกสิทธิ์การใช้งานสำหรับ user แต่ละประเภท

I. Lombok

Lombok เป็น library และ plugin ที่มี annotation แทนการสร้าง getter, setters, constructors, toString และ equals methods ทำให้ไม่ต้องแม้ให้ใช้ IDE ในการสร้างแบบอัตโนมัติ โค้ดจะสั้นลงและอ่านง่ายขึ้น

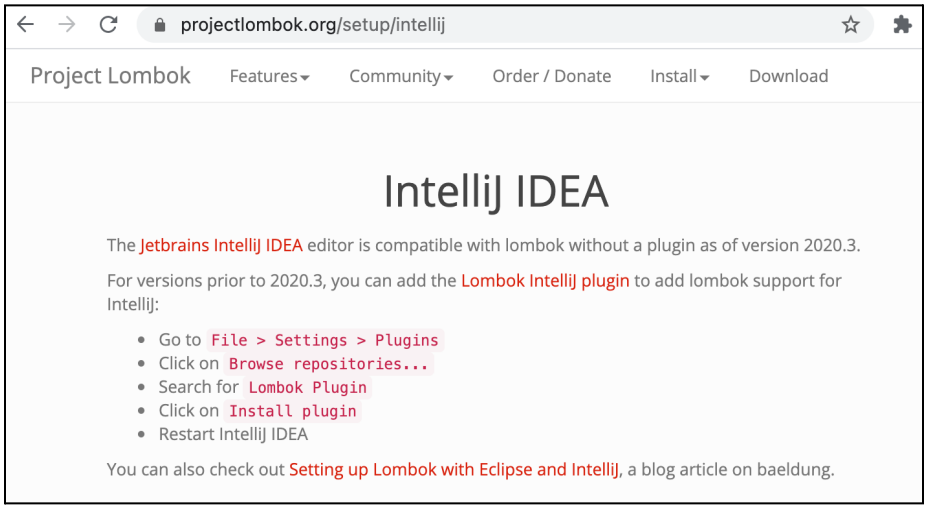
1. เพิ่ม Lombok dependencies (ถ้าเริ่มโปรเจคใหม่ สามารถเพิ่มได้ตั้งแต่ใน https://start.spring.io/)

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
```

2. ปรับ build config ที่อยู่ด้านล่าง ๆ ของ pom.xml เพื่อไม่ใช้ lombok ตอน build

```
<build>
  <plugins>
     <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <configuration>
           <excludes>
              <exclude>
                 <groupId>org.projectlombok</groupId>
                 <artifactId>lombok</artifactId>
              </exclude>
           </excludes>
        </configuration>
     </plugin>
  </plugins>
</build>
```

3. ให้ update IntelliJ เป็นรุ่นใหม่ (หลังเวอร์ชัน 2020.3)
    - ถ้ายังใช้รุ่นเก่า (ก่อนเวอร์ชัน 2020.3) ให้ติดตั้งI Lombok IntelliJ plugin ตามลิงก์ด้านล่าง
    - https://projectlombok.org/setup/intellij



4. ปรับแก้คลาส User ใน model package โดยเอา getters/setters ทั้งหมดออก และเพิ่ม **Lombok** annotations ดังนี้
    - `@Data` : เพิ่ม getters สำหรับทุก fields เพิ่ม setters สำหรับทุก fields ที่ไม่เป็น final และเพิ่ม toString, equals and hashCode
    - `@NoArgsConstructor` : สร้าง constructor ที่ไม่มี arguments

```
package th.ac.ku.restaurant.model;

import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import java.util.UUID;

@Data
@NoArgsConstructor
@Entity
public class User {

    @Id
    @GeneratedValue
    private UUID id;

    private String username;
    private String password;
    private String firstName;
    private String lastName;

    private String role;
}
```

II. DTO : Separating Transmission Data (DTO) and Database Data (DAO)

เนื่องจากเราส่ง object จาก database โดยตรงไปที่ client และรับข้อมูลจาก client มา save เข้า database โดยตรง ซึ่ง

- ไม่ค่อยปลอดภัย
- ไม่ค่อย efficient หากต้องการส่งข้อมูลของหลาย object แต่แต่ละ object ต้องการส่งแค่ 1-2 fields ทำให้ส่งหลาย object ไปให้ client

ดังนั้น เราควรแยก object ที่ใช้ส่งให้/รับจาก client ออกจาก object ที่จะ save ลง database ดังนี้

- **DAO (Data Access Object)** entity objects ที่เก็บลง database
- **DTO (Data Transfer Object)** objects ที่ส่งระหว่าง clients กับ servers
    - เราอาจอยากยิ่งกว่านี้ โดยแยก Request object ออกจาก Response object ด้วย

โดยเราสามารถกำหนดให้ DTO มี attributes ที่ต้องส่งให้ user เท่านั้น เช่น เราอาจไม่ต้องส่ง id ไปให้ client และ DTO อาจมี attributes จากหลาย entity objects ได้ ทำให้ส่งแค่ object เดียวก็พอได้

การ map ระหว่าง DAO และ DTO เราสามารถใช้ model mapper library มาช่วยได้

1. เพิ่ม Model Mapper dependencies

```
<dependency>
  <groupId>org.modelmapper</groupId>
  <artifactId>modelmapper</artifactId>
  <version>3.0.0</version>
</dependency>
```

2. สร้าง package config
3. สร้างคลาส ComponentConfig ใน package config
    - สร้างเมธอดเพื่อสร้าง Bean ModelMapper

```
package th.ac.ku.restaurant.config;

import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ComponentConfig {

   @Bean
   public ModelMapper modelMapper() {
       return new ModelMapper();
   }
}
```

4. เพิ่ม package **dto**
5. เพิ่มคลาส SignupDto ใน **dto** package สังเกตว่า
    - คลาสนี้ไม่ต้องมี id attribute เนื่องจากไม่ต้องใช้ในการส่งจาก client มาที่ server
    - คลาสนี้ไม่มี `@Entity` annotation
    - (นอกจากนั้น เราอาจแยกเพิ่มเติม ระหว่าง request objects และ response objects)

```
package th.ac.ku.restaurant.dto;

import lombok.Data;

@Data
public class SignupDto {
   private String username;
   private String password;
   private String firstName;
   private String lastName;
   private String role;
}
```

6. ปรับแก้ SignupController ให้ใช้คลาส SignupDto แทน User

```
package th.ac.ku.restaurant.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import th.ac.ku.restaurant.dto.SignupDto;
import th.ac.ku.restaurant.service.SignupService;

@Controller
public class SignupController {

   @Autowired
   private SignupService signupService;

   @GetMapping("/signup")
   public String getSignupPage() {
       return "signup"; // return signup.html
   }

   @PostMapping("/signup")
   public String signupUser(@ModelAttribute SignupDto user, Model model) {

       if (signupService.isUsernameAvailable(user.getUsername())) {
           signupService.createUser(user);
           model.addAttribute("signupSuccess", true);
       } else {
           model.addAttribute("signupError", "Username not available");
       }
       return "signup";
   }
}
```

7. ปรับแก้ UserService เพื่อ map ระหว่างคลาส SignupDto กับ User ก่อน save ลง database

```
package th.ac.ku.restaurant.service;

import org.modelmapper.ModelMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import th.ac.ku.restaurant.dto.SignupDto;
import th.ac.ku.restaurant.model.User;
import th.ac.ku.restaurant.repository.UserRepository;

@Service
public class SignupService {

   @Autowired
   private UserRepository repository;

   @Autowired
   private PasswordEncoder passwordEncoder;

   @Autowired
   private ModelMapper modelMapper;

   public boolean isUsernameAvailable(String username) {
       return repository.findByUsername(username) == null;
   }

   public void createUser(SignupDto user) {
       User record = modelMapper.map(user, User.class);
       record.setFirstName(user.getFirstName());
       record.setLastName(user.getLastName());
       record.setRole(user.getRole());
       record.setUsername(user.getUsername());

       String hashedPassword = passwordEncoder.encode(user.getPassword());
       record.setPassword(hashedPassword);

       repository.save(record);
   }

   public User getUser(String username) {
       return repository.findByUsername(username);
   }
}
```

8. ลองรันโปรแกรม เพื่อตรวจสอบว่า ยังทำงานได้เหมือนเดิมหรือไม่ http://localhost:8091/
   - ถ้าใช้ command line ใช้คำสั่ง

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

III. Input Validation

9. Spring Boot มี library ที่ช่วยตรวจสอบ input
    - เพิ่ม Spring Boot validation dependencies

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

10. เพิ่ม validation annotation ไปที่ตลาส dto
    - ตรวจสอบข้อมูลได้หลายแบบ
        - `@NotBlank` --- ตรวจสอบว่า ไม่ empty และไม่เป็น blank characters
        - `@NotEmpty` — ตรวจสอบว่า empty หรือไม่ เช่น ต้องไม่ใช่ `""`.
        - `@NotNull` --- ต้องไม่ null
    - `@Size` ตรวจสอบขนาด
    - `message` ใช้กำหนด error message
    - นอกจากนั้น ยังมี annotation อื่น ๆ ที่ช่วยในการตรวจสอบ input อีกมากมาย สามารถไปค้นคว้าเพิ่มเติมเองได้

```
package th.ac.ku.restaurant.dto;

import lombok.Data;

import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Pattern;
import javax.validation.constraints.Size;

@Data
public class SignupDto {

   @NotBlank
   @Size(min=4, message = "Username must have at least 4 characters")
   private String username;

   @NotBlank
   @Size(min=8, max=128, message = "Password must have at least 8 characters")
   private String password;

   @NotBlank(message = "First name is required")
   private String firstName;

   @NotBlank
   private String lastName;

   @NotBlank
   @Pattern(regexp = "^(ROLE_ADMIN|ROLE_USER)$",
            message = "Role is in an incorrect format.")
   private String role;
}
```

11. ปรับแก้ SignupController เพื่อตรวจสอบ input โดยเพิ่มโค้ดส่วน highlight สีเหลือง
    - `BindingResult` object ต้องอยู่หลังและติดกับ `@Valid` object.
    - ต้องส่ง SignupDto object ไปใน `@GetMapping` ด้วย
    - และต้องส่ง new SignupDto หลัง signup สำเร็จ

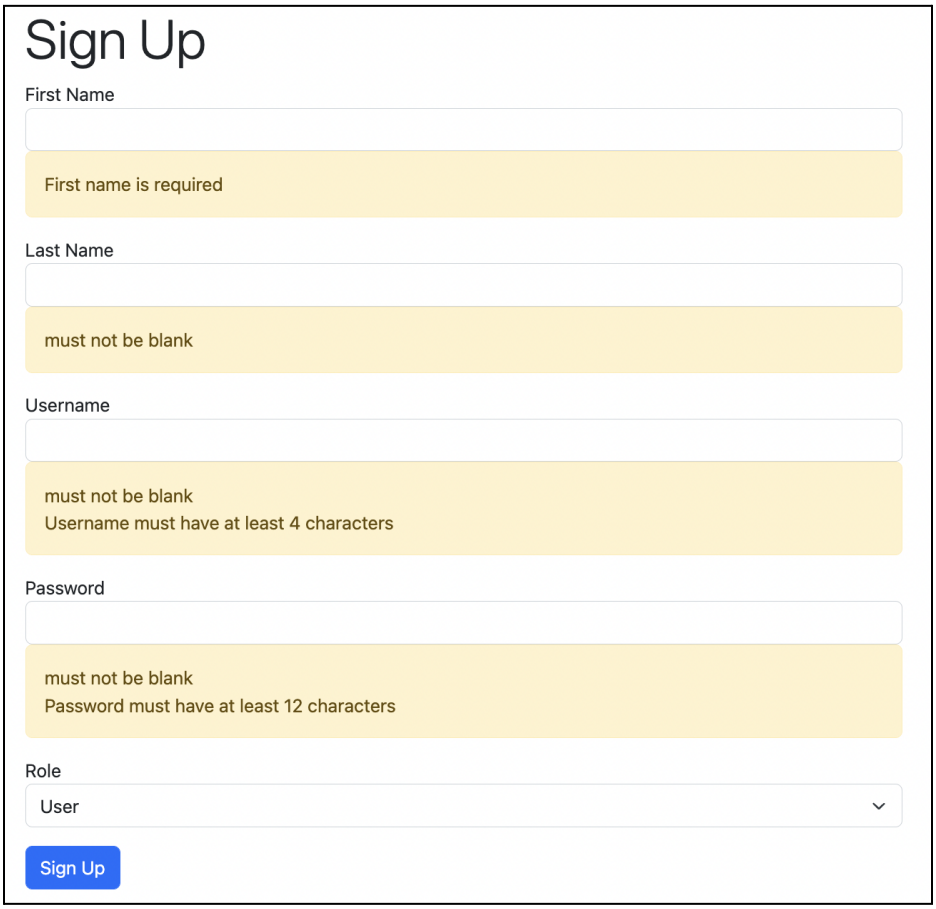
```
package th.ac.ku.restaurant.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import th.ac.ku.restaurant.dto.SignupDto;
import th.ac.ku.restaurant.service.SignupService;

import javax.validation.Valid;

@Controller
public class SignupController {

   @Autowired
   private SignupService signupService;

   @GetMapping("/signup")
   public String getSignupPage(SignupDto user) {
       return "signup"; // return signup.html
   }

   @PostMapping("/signup")
   public String signupUser(@Valid SignupDto user, BindingResult result,
                            Model model) {
       if (result.hasErrors())
           return "signup";

       if (signupService.isUsernameAvailable(user.getUsername())) {
           signupService.createUser(user);
           model.addAttribute("signupSuccess", true);
       } else {
           model.addAttribute("signupError", "Username not available");
       }
       model.addAttribute("signupDto", new SignupDto());
       return "signup";
   }
}
```

12. ปรับ signup.html เพื่อให้แสดง validation error messages

```
<!DOCTYPE html>
<html lang="en" xmlns:th="https://www.thymeleaf.org">
<head>
 <meta charset="UTF-8">
 <title>Restaurant Web Application</title>
 <link th:rel="stylesheet" th:href="@{/css/bootstrap.min.css}">
 <script th:src="@{/js/bootstrap.min.js}"></script>
</head>
<body>

<div th:insert="fragments/header :: header"></div>

<div class="container w-50 p-3">
 <h1 class="display-5">Sign Up</h1>

 <form action="#" th:action="@{/signup}" th:object="${signupDto}"
       method="POST">

   <div id="success-msg" class="alert alert-success"
        th:if="${signupSuccess}">
     Successfully signed up! Please <a th:href="@{/login}">login</a>.
   </div>
   <div id="error-msg" class="alert alert-danger"
        th:if="${signupError}">
     <span th:text="${signupError}"></span>
   </div>

   <div class="mb-3">
     <label for="inputFirstName">First Name</label>
     <input id="inputFirstName" type="input" type="text" class="form-control"
            name="firstName" th:field="*{firstName}">
     <div class="alert alert-warning"
          th:if="${#fields.hasErrors('firstName')}"
              th:errors="*{firstName}"></div>
   </div>
   <div class="mb-3">
     <label for="inputLastName">Last Name</label>
     <input id="inputLastName" type="input" type="text" class="form-control"
            name="lastName" th:field="*{lastName}">
     <div class="alert alert-warning"
          th:if="${#fields.hasErrors('lastName')}"
              th:errors="*{lastName}"></div>
   </div>
   <div class="mb-3">
     <label for="inputUsername">Username</label>
     <input id="inputUsername" type="input" type="text" class="form-control"
            name="username" th:field="*{username}">
     <div class="alert alert-warning"
          th:if="${#fields.hasErrors('username')}"
              th:errors="*{username}"></div>
   </div>
   <div class="mb-3">
     <label for="inputPassword">Password</label>
     <input id="inputPassword" type="password" class="form-control"
            name="password" th:field="*{password}">
     <div class="alert alert-warning"
          th:if="${#fields.hasErrors('password')}"
```

```
            th:errors="*{password}"></div>
    </div>
    <div class="mb-3">
     <label for="inputRole">Role</label>
     <select id="inputRole" class="form-select" name="role" th:field="*{role}">
       <option value="ROLE_USER" selected>User</option>
       <option value="ROLE_ADMIN">Admin</option>
     </select>
      <div class="alert alert-warning"
           th:if="${#fields.hasErrors('role')}"
               th:errors="*{role}"></div>
    </div>

    <button id="submit-button" type="submit" class="btn btn-primary">Sign
Up</button>
  </form>
</div>
</body>
</html>
```

13. รันโปรแกรมและลอง signup ด้วยข้อมูลที่ไม่ถูกต้อง จะเห็น error message



Input validation เพิ่มเติม

14. เพิ่มอีเมลใน user data ทั้งใน DTO และ DAO
    - และตรวจสอบ string pattern ในอีเมล

```
package th.ac.ku.restaurant.dto;

import lombok.Data;

import javax.validation.constraints.Email;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Pattern;
import javax.validation.constraints.Size;

@Data
public class SignupDto {

   @NotBlank
   @Size(min=4, message = "Username must have at least 4 characters")
   private String username;

   @NotBlank
   @Size(min=12, max=128, message = "Password must have at least 12
characters")
   private String password;

   @NotBlank(message = "First name is required")
   private String firstName;

   @NotBlank
   private String lastName;

   @NotBlank
   @Pattern(regexp = "^(ROLE_ADMIN|ROLE_USER)$",
           message = "Role is in an incorrect format.")
   private String role;

   @Email
   @NotBlank
   private String email;
}
```

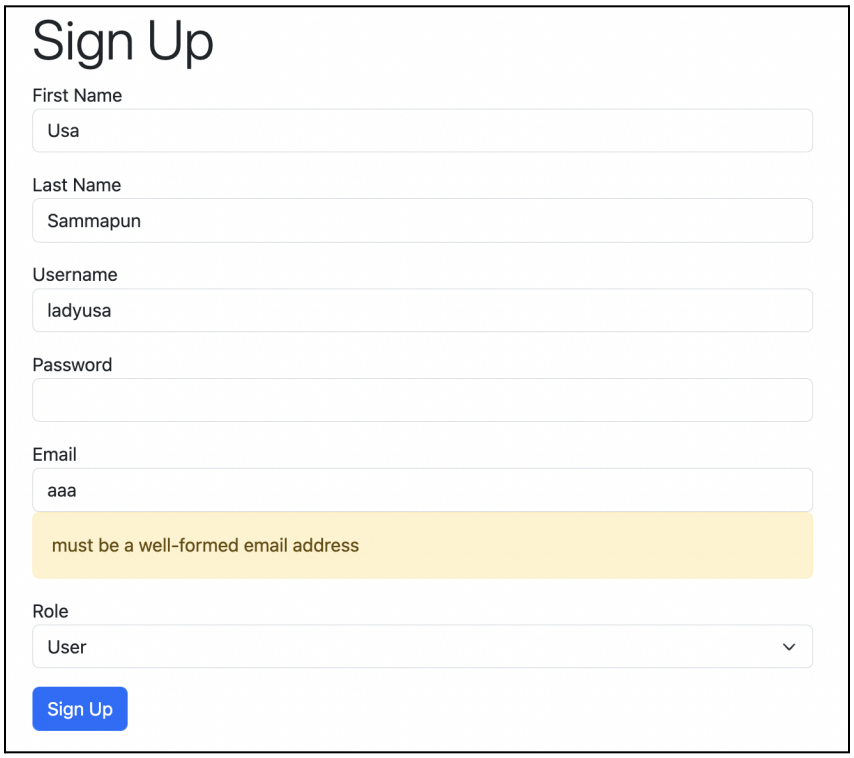
```
package th.ac.ku.restaurant.model;

import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import java.util.UUID;

@Data
@NoArgsConstructor
@Entity
```

```java
public class User {

    @Id
    @GeneratedValue
    private UUID id;

    private String username;
    private String password;
    private String firstName;
    private String lastName;
    private String email;

    private String role;
}
```

15. เพิ่มอีเมลใน signup form

```html
<div class="mb-3">
    <label for="inputEmail">Email</label>
    <input id="inputEmail" type="text" class="form-control"
           th:field="*{email}" >
    <div class="alert alert-warning" th:if="${#fields.hasErrors('email')}"
         th:errors="*{email}"></div>
</div>
```

16. รันโปรแกรม และไปที่หน้าเว็บ http://localhost:8091/
    - ลอง signup ด้วยข้อมูลอีเมลที่ไม่ถูกต้อง จะเห็น error message



IV. เชื่อมต่อไปที่ API

ขอ JWT Token

1. เพิ่ม Auth0 config ใน **application.properties**
    - ใช้ค่าเดียวกับ menu API

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

เช่น (ตรง uri บรรทัดสุดท้าย ต้องมี / ต่อท้ายด้วย)

```
auth0.audience=https://menu/api
auth0.clientId=TtEIf5Uu4TrhOasdfh0m6qD20XOY2sdfvUvxmm1VE
auth0.clientSecret=5vraLd6CW57EaszsdfMLa9XhRUDPBsfahWwdZr7-rNo0iS2sfQIkPfndfd4VE36N5CO0vB6Bv1m
spring.security.oauth2.resourceserver.jwt.issuer-uri=https://usa.jp.auth0.com/
```

2. สร้างคลาส JwtResponse ใน security package เพื่อเก็บ jwt response JSON object

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

import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.Data;

@Data
public class JwtResponse {

   @JsonProperty("access_token")
   private String accessToken;

   @JsonProperty("expires_in")
   private int expiresIn;

   @JsonProperty("token_type")
   private String tokenType;
}
```

3. เพิ่มการสร้าง Bean RestTemplate ในคลาส ComponentConfig ใน package config
    - RestTempate เป็นคลาสที่ใช้ในการเชื่อมต่อไปที่ API

```
package th.ac.ku.restaurant.config;

import org.modelmapper.ModelMapper;
import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
import org.springframework.web.client.RestTemplate;

@Configuration
public class ComponentConfig {

    @Bean
    public ModelMapper modelMapper() {
        return new ModelMapper();
    }

    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder builder) {
        return builder.build();
    }
}
```

4. สร้างคลาส JwtAccessTokenService ใน package security
    - เพื่อขอ Jwt token จาก Auth0 และเก็บ token เอาไว้ใช้

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

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.*;
import org.springframework.stereotype.Service;
import org.springframework.util.LinkedMultiValueMap;
import org.springframework.util.MultiValueMap;
import org.springframework.web.client.RestTemplate;

@Service
public class JwtAccessTokenService {

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

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

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

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

    @Autowired
    private RestTemplate restTemplate;

    private String token = null;

    public String requestAccessToken() {

        if (token != null)
            return token;

        HttpHeaders headers = new HttpHeaders();
```

```
        headers.add("Content-Type",
                    MediaType.APPLICATION_FORM_URLENCODED.toString());

        MultiValueMap<String, String> requestBody =
                                              new LinkedMultiValueMap<>();
        requestBody.add("grant_type", "client_credentials");
        requestBody.add("client_id", clientId);
        requestBody.add("client_secret", clientSecret);
        requestBody.add("audience", audience);

        HttpEntity entity = new HttpEntity(requestBody, headers);

        ResponseEntity<JwtResponse> response =
                restTemplate.exchange(issuer + "oauth/token",
                                      HttpMethod.POST,
                                      entity, JwtResponse.class);

        JwtResponse jwtResponse = response.getBody();
        token = jwtResponse.getAccessToken();

        return token;
    }
}
```

GET Menu จาก API

5. สร้างคลาส MenuDto ใน package dto เพื่อรองรับ response จาก API

```
package th.ac.ku.restaurant.dto;

import lombok.Data;

import java.util.UUID;

@Data
public class MenuDto {
    private UUID id;
    private String name;
    private double price;
    private String category;
}
```

6. สร้างคลาส MenuService ใน package service
    - ซึ่งจะส่ง GET request ไปพร้อมกับ Jwt Token
    - **การ GET** จะใช้เมทอด getForEntity ใน RestTemplate
    - ซึ่งเราจะได้เป็น ResponseEntity
        - ในนี้จะมี status code แจ้งว่า การเชื่อมต่อโอเคมั้ย
        - และจะมี response body เป็น array ของ Restaurant
    - จากนั้น เราจะต้องเรียก getBody() เพื่อให้ได้ array นี้

- สุดท้าย เราจะแปลง array เป็นลิสต์แล้วคืนค่ากลับไปให้ controller

```
package th.ac.ku.restaurant.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;
import th.ac.ku.restaurant.dto.MenuDto;
import th.ac.ku.restaurant.security.JwtAccessTokenService;

import java.util.Arrays;
import java.util.List;

@Service
public class MenuService {

   @Autowired
   private RestTemplate restTemplate;

   @Autowired
   private JwtAccessTokenService tokenService;

   public List<MenuDto> getMenus() {

       String token = tokenService.requestAccessToken();

       HttpHeaders headers = new HttpHeaders();
       headers.add("authorization", "Bearer " + token);
       HttpEntity entity = new HttpEntity(headers);

       String url = "http://localhost:8090/menu";

       ResponseEntity<MenuDto[]> response =
               restTemplate.exchange(url, HttpMethod.GET,
                       entity, MenuDto[].class);

       MenuDto[] menus = response.getBody();
       return Arrays.asList(menus);
   }
}
```

7. สร้างคลาส MenuController ใน package controller

```
package th.ac.ku.restaurant.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import th.ac.ku.restaurant.service.MenuService;

@Controller
@RequestMapping("/menu")
public class MenuController {

   @Autowired
   private MenuService service;

   @GetMapping
   public String getMenus(Model model) {
       model.addAttribute("menus", service.getMenus());
       return "menu";
   }
}
```

8. สร้างหน้า menu.html ใน template

```
<!DOCTYPE html>
<html lang="en" xmlns:th="https://www.thymeleaf.org">
<head>
 <meta charset="UTF-8">
 <title>Restaurant Web Application</title>
 <link th:rel="stylesheet" th:href="@{/css/bootstrap.min.css}">
 <script th:src="@{/js/bootstrap.min.js}"></script>
</head>
<body>

<div th:insert="fragments/header :: header"></div>

<div class="container">
 <h1 class="display-5">Menus</h1>

 <table class="table table-striped">
   <thead>
   <tr>
     <th>Name</th>
     <th>Price</th>
     <th>Category</th>
   </tr>
   </thead>
   <tbody>
   <tr th:each="menu : ${menus}">
     <td th:text="${menu.name}"></td>
     <td th:text="${menu.price}"></td>
```

```
      <td th:text="${menu.category}"></td>
    </tr>
    </tbody>
  </table>
</div>

</body>
</html>
```

POST Menu ไปที่ API

9. เพิ่มการ POST ใน MenuService เพื่อส่งการ add menu ไปที่ Menu API

```
package th.ac.ku.restaurant.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.*;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;
import th.ac.ku.restaurant.dto.MenuDto;
import th.ac.ku.restaurant.security.JwtAccessTokenService;

import java.util.Arrays;
import java.util.List;

@Service
public class MenuService {

    @Autowired
    private RestTemplate restTemplate;

    @Autowired
    private JwtAccessTokenService tokenService;

    public List<MenuDto> getMenus() {

        String token = tokenService.requestAccessToken();

        HttpHeaders headers = new HttpHeaders();
        headers.add("authorization", "Bearer " + token);
        HttpEntity entity = new HttpEntity(headers);

        String url = "http://localhost:8090/menu";

        ResponseEntity<MenuDto[]> response =
                restTemplate.exchange(url, HttpMethod.GET,
                        entity, MenuDto[].class);

        MenuDto[] menus = response.getBody();
        return Arrays.asList(menus);
    }
```

```
    public MenuDto addMenu(MenuDto menu) {

        String token = tokenService.requestAccessToken();

        HttpHeaders headers = new HttpHeaders();
        headers.add("authorization", "Bearer " + token);
        headers.add("Content-Type", MediaType.APPLICATION_JSON.toString());
        HttpEntity entity = new HttpEntity(menu,headers);

        String url = "http://localhost:8090/menu";

        ResponseEntity<MenuDto> response =
                restTemplate.exchange(url, HttpMethod.POST,
                        entity, MenuDto.class);

        return response.getBody();
    }

}
```

10. ปรับ MenuController เพื่อเพิ่มการ POST

```
package th.ac.ku.restaurant.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import th.ac.ku.restaurant.dto.MenuDto;
import th.ac.ku.restaurant.service.MenuService;

import javax.validation.Valid;

@Controller
@RequestMapping("/menu")
public class MenuController {

    @Autowired
    private MenuService service;

    @GetMapping
    public String getMenus(Model model) {
        model.addAttribute("menus", service.getMenus());
        return "menu";
    }

    @GetMapping("/add")
    public String getMenuForm(MenuDto menuDto) {
        return "menu-add";
    }
```

```
    @PostMapping("/add")
    public String addMenu(@Valid MenuDto menuDto, BindingResult result,
                          Model model) {
        if (result.hasErrors())
            return "menu-add";

        service.addMenu(menuDto);
        return "redirect:/menu";
    }
}
```

11. เพิ่มหน้าฟอร์มเพิ่ม menu ใน menu-add.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="https://www.thymeleaf.org">
<head>
 <meta charset="UTF-8">
 <title>Restaurant Web Application</title>
 <link th:rel="stylesheet" th:href="@{/css/bootstrap.min.css}">
 <script th:src="@{/js/bootstrap.min.js}"></script>
</head>
<body>

<div th:insert="fragments/header :: header"></div>

<div class="container w-50 p-3">
 <h1 class="display-5">Add Menu</h1>

 <form action="#" th:action="@{/menu/add}" th:object="${menuDto}"
method="POST">

   <div class="mb-3">
     <label for="inputName">Name</label>
     <input id="inputName" type="text" class="form-control"
            th:field="*{name}" >
     <div class="alert alert-warning" th:if="${#fields.hasErrors('name')}"
          th:errors="*{name}"></div>
   </div>
   <div class="mb-3">
     <label for="inputPrice">Price</label>
     <input id="inputPrice" type="text" class="form-control"
            th:field="*{price}">
     <div class="alert alert-warning" th:if="${#fields.hasErrors('price')}"
          th:errors="*{price}"></div>
   </div>
   <div class="mb-3">
     <label for="inputCategory">Category</label>
     <select id="inputCategory" class="form-select" th:field="*{category}">
       <option value="Appetizer" selected>Appetizer</option>
       <option value="Main Course">Main Course</option>
       <option value="Dessert">Dessert</option>
     </select>
     <div class="alert alert-warning"
          th:if="${#fields.hasErrors('category')}"
          th:errors="*{category}"></div>
```

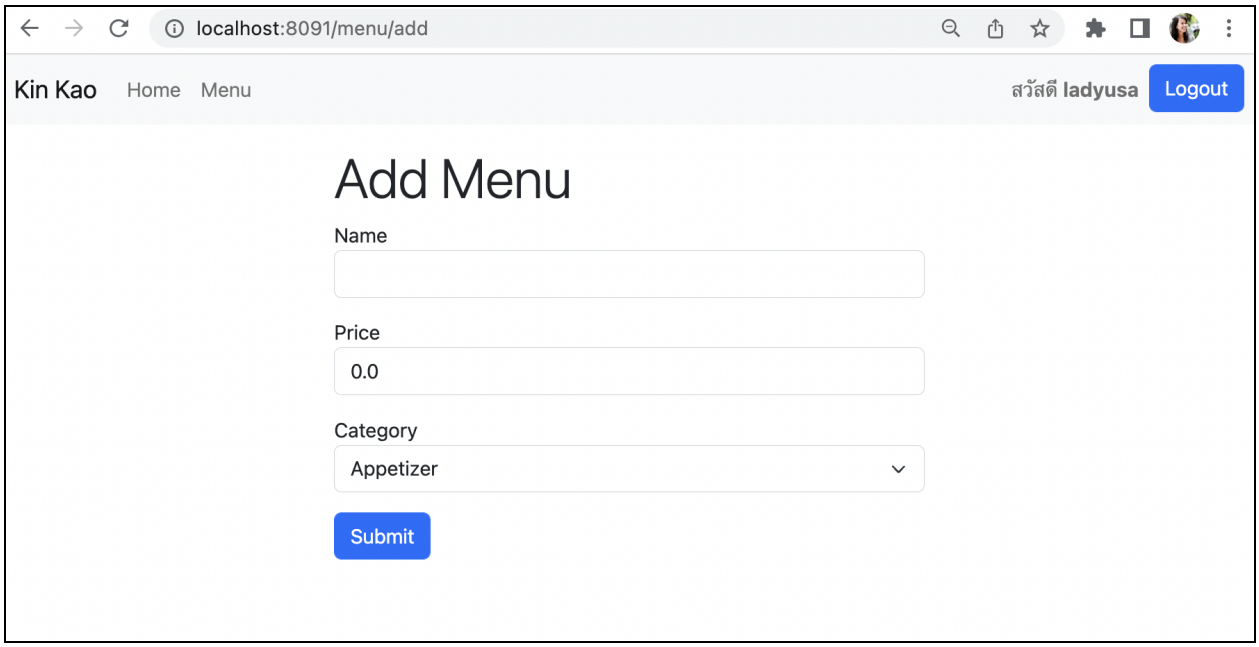
```
    </div>
    <button id="submit-button" type="submit" class="btn
btn-primary">Submit</button>
  </form>
</div>

</body>
</html>
```

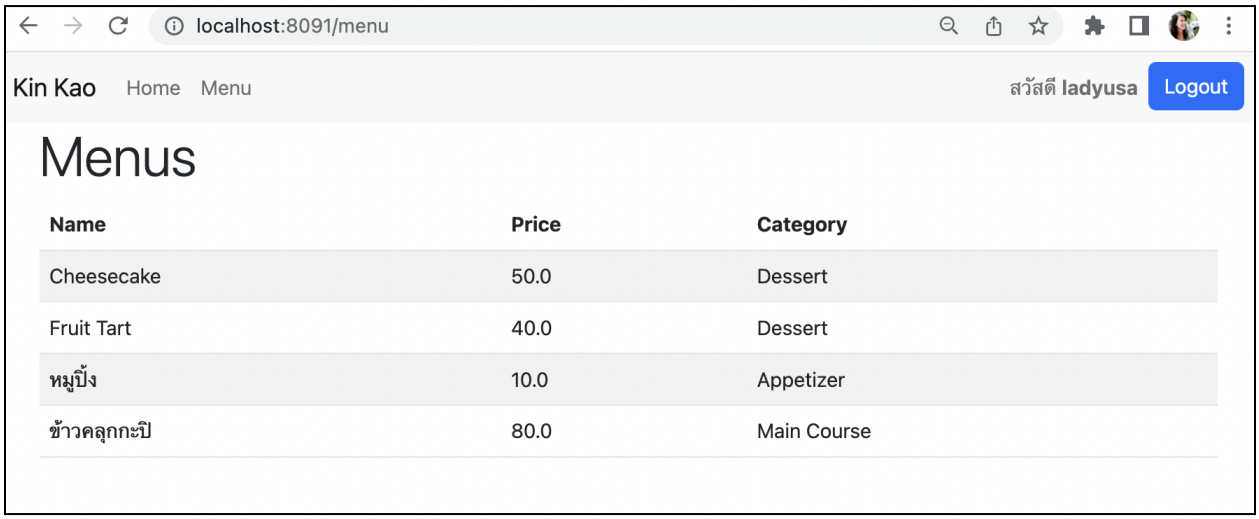
12. เพิ่ม validation ใน MenuDto
    - Validation นี้ใช้กับ html template เท่านั้น ไม่ได้ใช้เมื่อรับข้อมูลจาก Menu API

```
package th.ac.ku.restaurant.dto;

import lombok.Data;

import javax.validation.constraints.Min;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;
import java.util.UUID;

@Data
public class MenuDto {

    private UUID id;

    @NotBlank
    private String name;

    @NotNull
    @Min(value = 0)
    private double price;

    @NotBlank
    private String category;
}
```

13. รันโปรแกรม Menu API และ database ของ Menu เตรียมเอาไว้

14. ลองรันโปรแกรม และไปที่หน้าเว็บ http://localhost:8091/
    - signup, login
    - ไปที่ http://localhost:8091/menu/add
    - จะเห็นหน้าฟอร์ม ให้เพิ่มรายการอาหาร
    - เมื่อกดปุ่ม submit จะเห็นข้อมูลใหม่ในตาราง menu





15. ลองตรวจสอบใน database ของ menu API ด้วยก็ได้ว่า มีข้อมูลใหม่เข้ามาจริงหรือไม่

V. แยกสิทธิ์การใช้งานระหว่าง admin และ user

1. ปรับแก้ header.html เพื่อให้ admin สามารถเข้าถึงการเพิ่ม menu ได้เท่านั้น user จะไม่เห็น

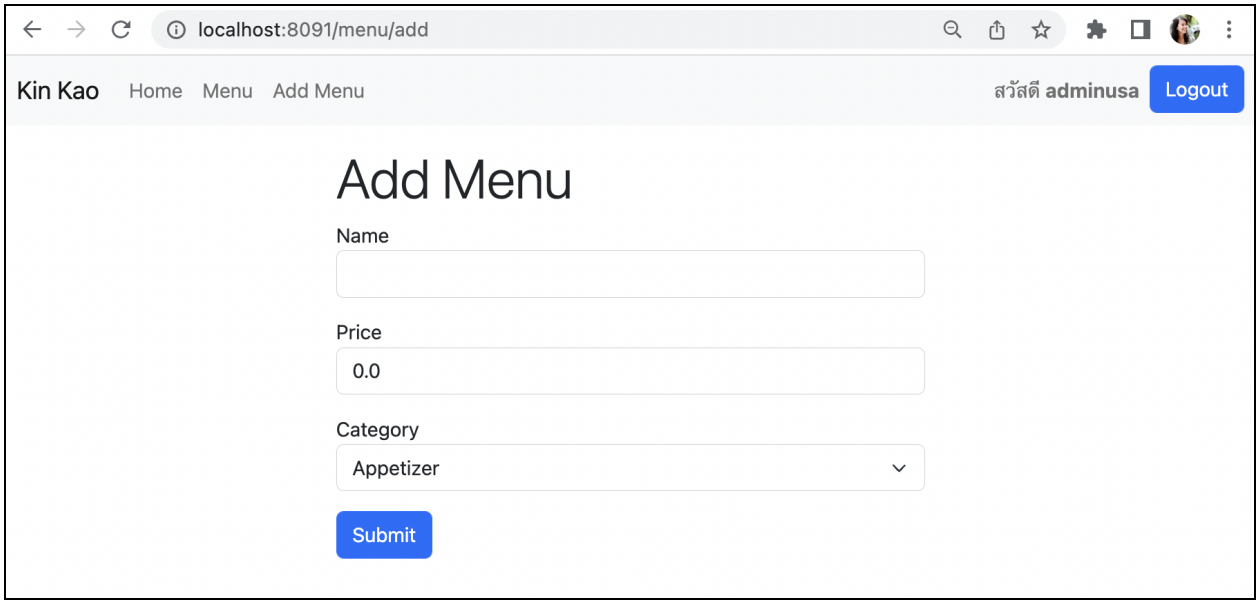
```
<!DOCTYPE html>
<html lang="en" xmlns:th="https://www.thymeleaf.org"
      xmlns:sec="https://www.thymeleaf.org/thymeleaf-extras-springsecurity5">
<head>
 <meta charset="UTF-8">
 <link th:rel="stylesheet" th:href="@{/css/bootstrap.min.css}">
 <script th:src="@{/js/bootstrap.min.js}"></script>
</head>

<nav class="navbar navbar-expand-lg navbar-light bg-light"
th:fragment="header">

 <div class="container-fluid">
   <a class="navbar-brand" th:href="@{/}">Kin Kao</a>

   <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent"
aria-controls="navbarSupportedContent" aria-expanded="false"
aria-label="Toggle navigation">
     <span class="navbar-toggler-icon"></span>
   </button>

   <div class="collapse navbar-collapse" id="navbarSupportedContent">
     <ul class="navbar-nav me-auto mb-2 mb-lg-0">
       <li class="nav-item">
         <a class="nav-link" th:href="@{/}">Home</a>
       </li>
       <li class="nav-item">
         <a class="nav-link" th:href="@{/menu}">Menu</a>
       </li>
       <div sec:authorize="hasRole('ROLE_ADMIN')">
         <li class="nav-item">
           <a class="nav-link" th:href="@{/menu/add}">Add Menu</a>
         </li>
       </div>
     </ul>
   </div>
   <div sec:authorize="isAuthenticated()">
     <ul class="navbar-nav me-auto mb-2 mb-lg-0">
       <li class="nav-item">
         <a class="nav-link" href="#">สวัสดี <b><span
sec:authentication="name"></span></b></a>
       </li>
       <li class="nav-item">
         <a class="btn btn-primary" th:href="@{/logout}">Logout</a>
       </li>
     </ul>
   </div>
   <div sec:authorize="isAnonymous()">
     <ul class="navbar-nav me-auto mb-2 mb-lg-0">
       <li class="nav-item">
```

```
          <a class="btn btn-primary" th:href="@{/login}">Login</a>
        </li>
      </ul>
    </div>
  </div>
</nav>

</nav>
</body>
</html>
```

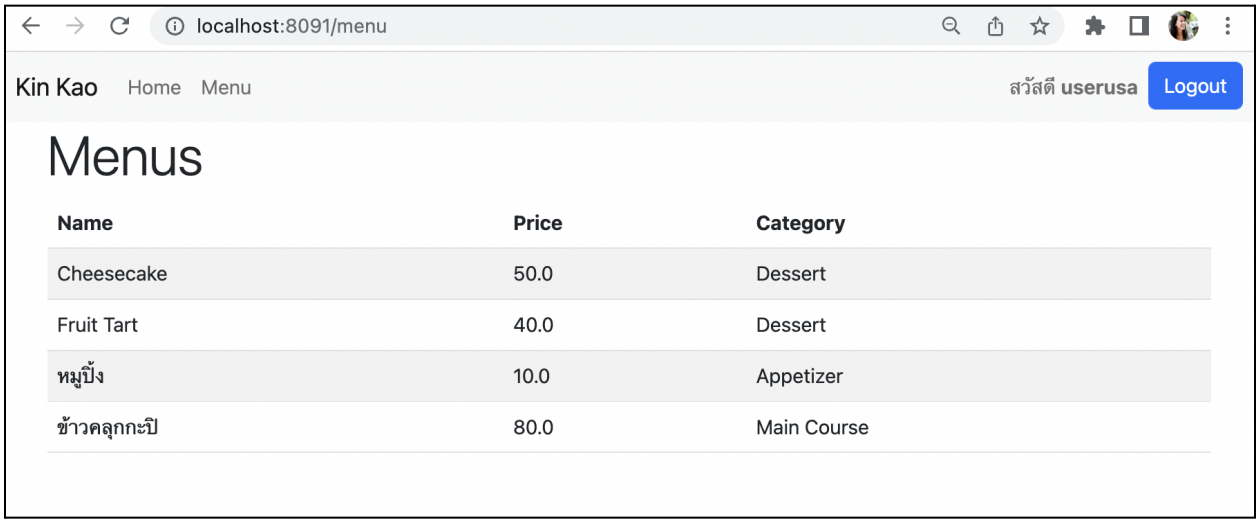
2. ปรับคลาส SecurityConfig เพื่อจำกัดการเข้าถึงหน้านั้น ๆ ด้วย

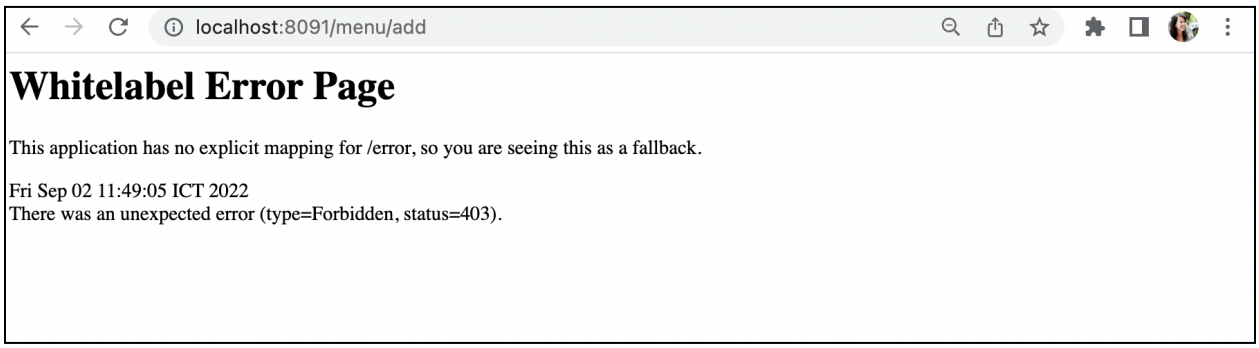
```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserDetailsServiceImp userDetailsService;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                .antMatchers("/", "/signup", "/css/**", "/js/**").permitAll()
                .antMatchers("/menu/add")
                    .access("hasRole('ROLE_ADMIN')")
                .antMatchers("/menu")
                    .access("hasRole('ROLE_USER') or hasRole('ROLE_ADMIN')")
                .anyRequest().authenticated()
        .and()
               // . . . existing code . . .

    }
    // . . . existing code . . .
}
```

3. รันโปรแกรม signup, login และไปที่หน้าเว็บ http://localhost:8091/
    - Signup โดยเลือกสิทธิ์ทั้ง 2 ประเภท
    - ลองเข้าหน้า Add Menu ทั้ง 2 ประเภท
        - Admin – เห็นเมนู Add Menu ใน nav bar และเข้าหน้าฟอร์มการเพิ่มได้



- User – ไม่เห็นเมนู และถ้าไปหน้า /menu/add โดยตรง จะได้ 403 Forbidden error





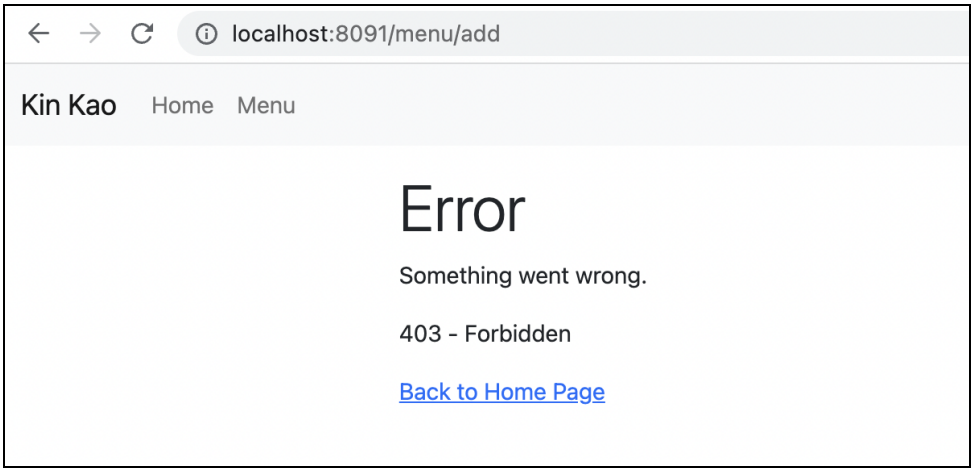
VI. Custom Error Page

1. ถ้าเกิด error เราจะได้หน้า default error page ซึ่งไม่ friendly ต่อผู้ใช้ และไม่ค่อยปลอดภัย
    - ในบางครั้ง หน้า default error page อาจแสดง error stack trace ดังรูปข้างล่าง ซึ่งทำให้ attacker รู้เลยว่า เราใช้ Spring Framework และไปหาช่องโหว่มา attack เราได้
    - ดังนั้น เราควรสร้างหน้า error page ของเราเองที่ไม่แสดง stack trace ในลักษณะนี้

2. ทำได้โดยสร้างหน้า “error.html” template และใส่ข้อมูลให้ user friendly และทั่วไปพอที่จะไม่ให้ attacker รู้ข้อมูลเกี่ยวกับ server ของเรา

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
   <meta charset="UTF-8">
   <title>Error</title>
   <link th:rel="stylesheet" th:href="@{/css/bootstrap.min.css}">
</head>

<body
th:with="httpStatus=${T(org.springframework.http.HttpStatus).valueOf(#response
.status)}">

<div th:insert="fragments/header :: header"></div>

<div class="container w-50 p-3">
   <h1 class="display-5">Error</h1>
   <p>Something went wrong.</p>
   <p th:text="|${#response.status} - ${httpStatus.reasonPhrase}|"></p>
   <p><a th:href="@{/}">Back to Home Page</a></p>
</div>

</body>
</html>
```

3. ลอง login เป็น user และไปหน้า add menu จะได้ error ที่ user friendly และปลอดภัยขึ้น


localhost:8091/menu/add
Kin Kao
Home
Menu
Error
Something went wrong.
403 - Forbidden
Back to Home Page