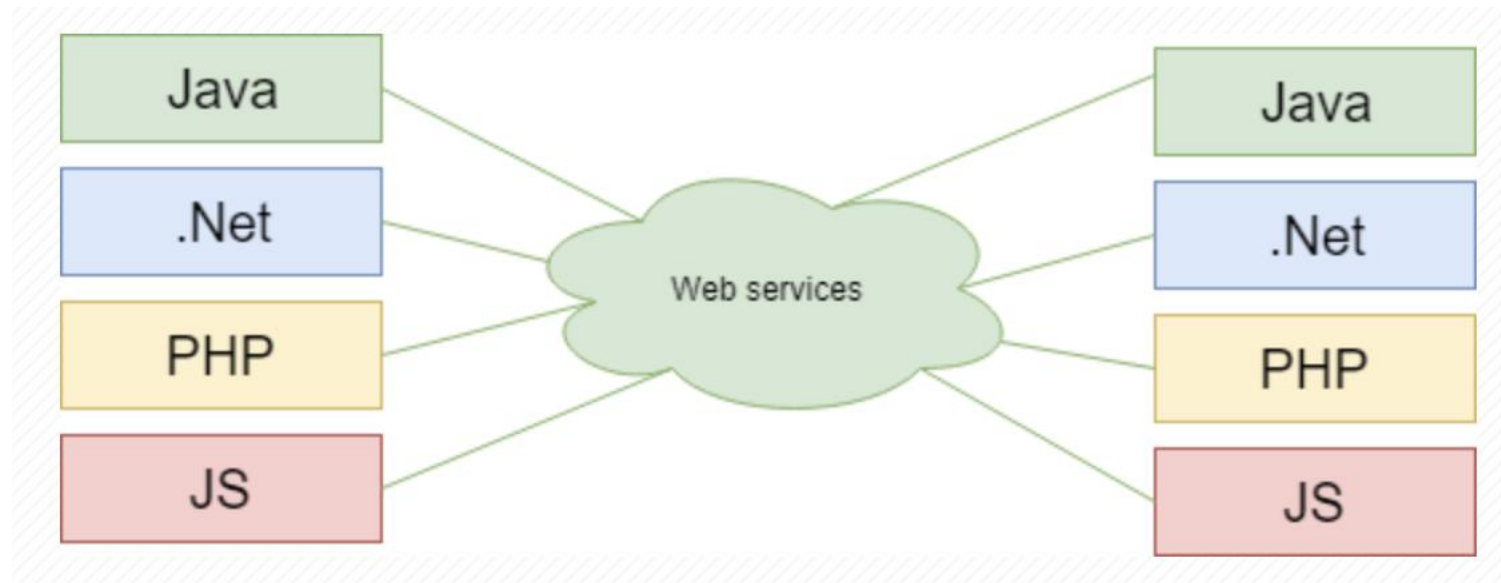


# **Web Services và Spring Rest API**

# Web Service

- Là tập hợp các giao thức và tiêu chuẩn mở được sử dụng để trao đổi dữ liệu giữa các ứng dụng qua mạng
- Có thể giúp cho các phần mềm được viết bằng các ngôn ngữ khác nhau hoặc trên các nền tảng khác nhau có thể trao đổi dữ liệu qua lại



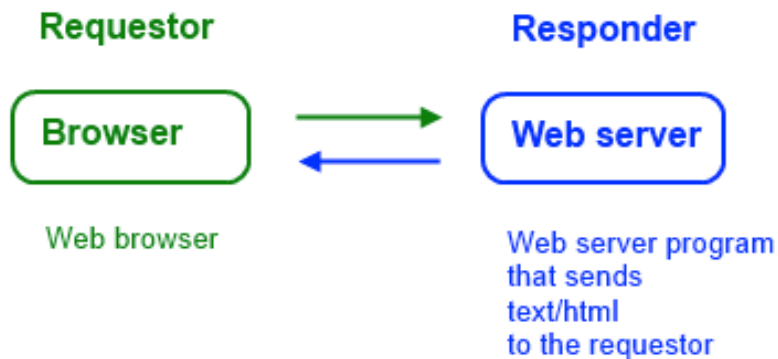
## Web Service vs Web Application

- Ứng dụng Web (Web App) dành cho người dùng và có thể truy cập qua trình duyệt, còn Web service dành cho ứng dụng truy cập thông qua các định dạng XML, Json ...
- Web App luôn sử dụng giao thức HTTP trong khi Web Service trước đây sử dụng giao thức SOAP (Simple Object Access Protocol) và gần đây sử dụng REST (Representational State Transfer).
- Web App không có khả năng tái sử dụng, trong khi Web Service có khả năng này và có thể được sử dụng bởi nhiều loại ứng dụng.

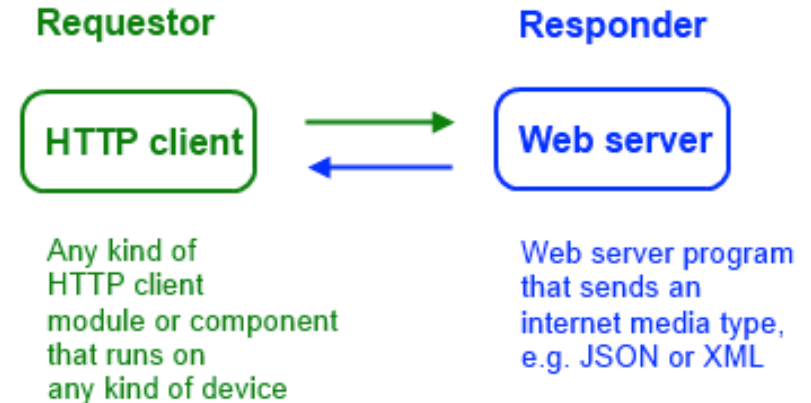
# Web Service vs Web Application

- Web App có thể truy cập Web Service để thu thập dữ liệu hoặc thực hiện tác vụ, nhưng Web Service không thể truy cập Web App.

## Web app:



## Web service:



## RESTful Web Service (API) trong Spring

- REST là một kiểu kiến trúc không phải là giao thức.
- REST không định nghĩa các định dạng truyền dữ liệu chuẩn, có thể xây dựng REST service với định dạng XML hoặc JSON (JSON hiện được ưa chuộng hơn).
- Một REST service có thể được yêu cầu bằng các phương thức HTTP như GET, POST, PUT, DELETE ....:
  - GET /users: Lấy danh sách người dùng
  - POST /user: Thêm mới người dùng
- Một số ưu điểm của REST:
  - Độc lập nền tảng
  - Hỗ trợ nhiều định dạng dữ liệu (XML, JSON, HTML ...)
  - Tốc độ nhanh hơn so với SOAP

## Tạo GET API để trả về danh sách thành phần

- Sửa từ Project Taco-cloud JPA (xoá các lớp controllers trong gói tacos.web và các templates)
- Tạo lớp Rest Controller là IngredientController trong package tacos.web.api

```
package tacos.web.api;  
  
import java.util.Optional;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.hateoas.server.EntityLinks;  
import org.springframework.web.bind.annotation.CrossOrigin;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import tacos.Ingredient;  
  
import tacos.data.IngredientRepository;
```

# Tạo GET API để trả về danh sách thành phần

```
@RestController
@RequestMapping(path = "/ingredients", produces = "application/json")
@CrossOrigin(origins = "*")
public class IngredientController {
    private IngredientRepository ingredientRepo;
    @Autowired
    EntityLinks entityLinks;
    public IngredientController(IngredientRepository ingredientRepo) {
        this.ingredientRepo = ingredientRepo;
    }
}
```

# Tạo GET API để trả về danh sách thành phần

```
@GetMapping
public Iterable<Ingredient> getAllIngredients() {
    return ingredientRepo.findAll();
}
```

```
@GetMapping("/{id}")
public Ingredient ingredientById(@PathVariable("id") String id) {
    Optional<Ingredient> optIngredient = ingredientRepo.findById(id);
    if (optIngredient.isPresent()) {
        return optIngredient.get();
    }
    return null;
}
}
```



## Tạo GET API để trả về danh sách thành phần

- `@RestController`: để đánh dấu lớp điều khiển cho Rest API
- Lớp `RestController` cũng được khởi tạo như 1 bean trong Spring, nhưng các phương thức xử lý request sẽ trả kết quả về thân response chứ không ghi vào Model và chuyển đến view như Controller thông thường.
- `@RequestMapping`: ấn định đường dẫn cho lớp tương tự các Controller khác, nhưng có thêm thuộc tính `produces` trả kết quả dạng Json (có thể ấn định loại khác)
- `@CrossOrigin`: Cho phép gọi API từ máy chủ khác localhost.

## Tạo GET API để trả về danh sách thành phần

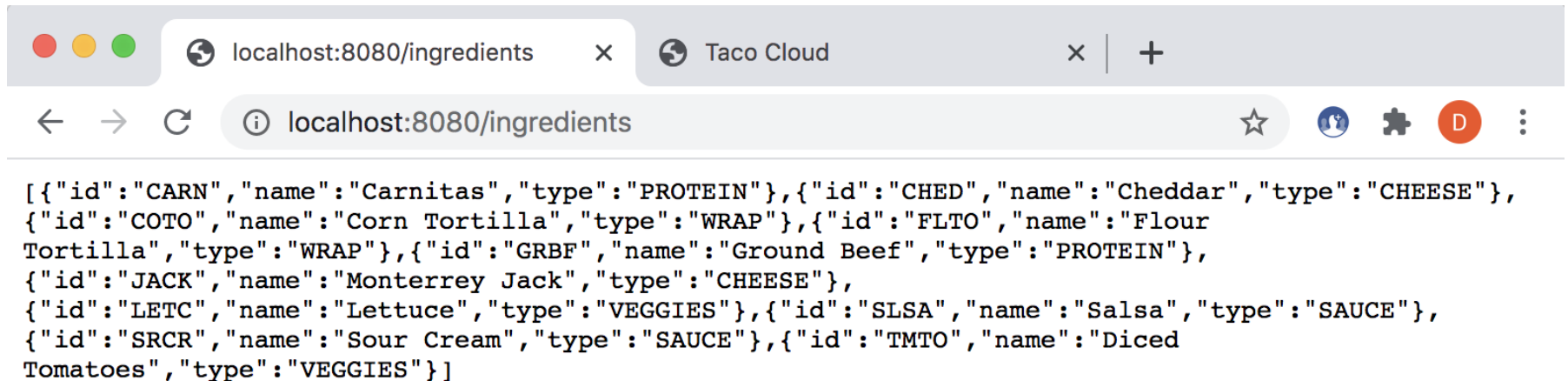
- Đối tượng IngredientRepository được gắn vào lớp này để tương tác với CSDL
- Phương thức getAllIngredients() được chú giải bằng @GetMapping đánh dấu nó sẽ được gọi xử lý GET request khi có lời gọi API tới controller. Phương thức này lấy và trả về danh sách các Ingredients nhờ Ingredient Repository.
- Chú giải @GetMapping cho phương thức này không có thêm tham số đường dẫn nên nó trùng với đường dẫn mức lớp

## Tạo GET API để trả về danh sách thành phần

- Phương thức `ingredientById()` sẽ tìm Ingredient theo Id và trả về kết quả
- `@GetMapping("/{id}")` cho biết id sẽ được gửi theo đường dẫn
- `@PathVariable("id")` cho biết id sẽ lấy từ đường dẫn và sau đó được sử dụng trong phương thức như một tham số

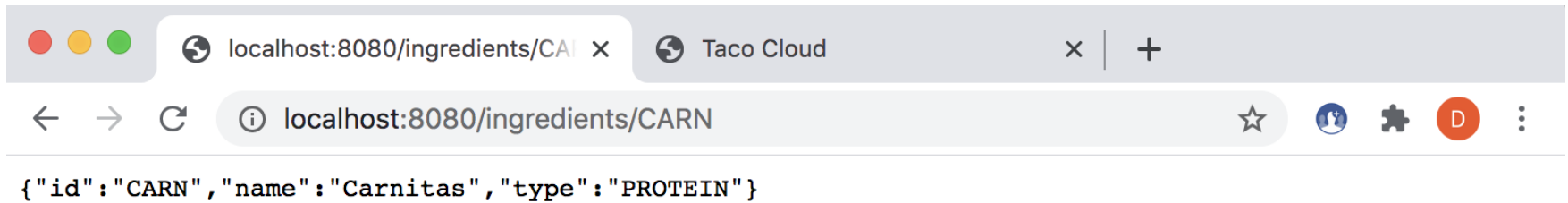
# Test nhanh GET API

- Bật trình duyệt gõ localhost:8008/ingredients:
  - Lưu ý chỉ áp dụng cho GET API. Nếu các API loại khác sẽ báo lỗi Method Not Allowed (phải dùng công cụ khác như Postman)



## Test nhanh GET API

- Để lấy 1 thành phần có mã là CARN gõ:  
localhost:8008/ingredients/CARN



## Sử dụng GET API vừa tạo

- Có thể gọi tới GET API vừa tạo từ bất kỳ ứng dụng HTTP Client nào (web/mobile ...).
- Nếu sử dụng các thư viện HTTP bậc thấp, phải tiến hành nhiều xử lý khi gọi API như tạo client instance và request object, thực thi request, trích xuất response, chuyển đổi về các đối tượng thực thể, v.v.
- Để gọi tới REST API trong Spring, có thể sử dụng thư viện RestTemplate.
- RestTemplate cung cấp các phương thức để gọi Rest API và xử lý các kết quả một cách tự động, giảm bớt các xử lý thủ công.

## Sử dụng GET API vừa tạo

- Tạo ứng dụng Spring Web độc lập với ứng dụng API Server trên để đóng vai trò của 1 Client App (frontend App)
- Sử dụng lại Project Taco-cloud phiên bản đầu tiên (chưa truy cập CSDL)
- Tiến hành gọi API vừa tạo để lấy danh sách thành phần từ Server thay vì hard code trong mã nguồn (hoặc trực tiếp truy cập CSDL để lấy như các project sau đó)
- Lưu ý đổi cổng Tomcat cho ứng dụng Client này để nó chạy đồng thời với ứng dụng API Server
  - Thêm dòng `server.port=8081` vào file `application.properties`

## Thực hiện một số thay đổi trên các lớp thực thể

- Bổ sung chú giải `@NoArgsConstructor` cho lớp `Ingredient`

```
@NoArgsConstructor(force=true)
```

```
public class Ingredient {
```

- Bổ sung các thuộc tính `id` và `createdAt` cho lớp `Taco`.

Thay đổi thuộc tính `ingredients` từ `List<String>` về `List<Ingredient>`

```
public class Taco {
```

```
    private Long id;
```

```
    private String name;
```

```
    private Date createdAt;
```

```
    private List<Ingredient> ingredients;
```

```
}
```

- Bổ sung thuộc tính `id` và `placedAt` cho lớp `Order`. Bổ sung thuộc tính `List<Taco> tacos`.



# Chỉnh sửa lớp TacoDesignController

```
public class DesignTacoController {  
    private RestTemplate rest = new RestTemplate();  
    @GetMapping  
    public String showDesignForm(Model model) {  
        List<Ingredient> ingredients =  
            Arrays.asList(rest.getForObject("http://localhost:8080/  
                ingredients", Ingredient[].class));  
        Type[] types = Ingredient.Type.values();  
        for (Type type : types) {  
            model.addAttribute(type.toString().toLowerCase(),  
                filterByType(ingredients, type));  
        }  
        model.addAttribute("design", new Taco());  
        return "design";  
    }  
}
```

...

## Chỉnh sửa lớp TacoDesignController

- Khai báo đối tượng RestTemplate trong lớp TacoDesignController.
- Sử dụng phương thức `getForObject(url, class)` để gọi API. Phương thức sẽ tự động gọi API, lấy kết quả, chuyển đổi thành mảng các đối tượng. Các tham số đáng chú ý:
  - url: đường dẫn của API
  - class: Lớp thực thể sẽ được dùng để chuyển đổi kết quả từ API sang
  - Trường hợp gọi API tìm kiếm theo ID có thể bổ sung tham số ID. Ví dụ:

```
rest.getForObject("http://localhost:8080/  
ingredients/{id}", Ingredient.class, ingredientId)
```

# Tạo POST API để lưu thông tin thiết kế bánh

- Quay lại Project API Server
- Tạo lớp TacoDesignController trong gói tacos.web.api

```
package tacos.web.api;

import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.hateoas.server.EntityLinks;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;
import tacos.Taco;
import tacos.data.TacoRepository;
```

# Tạo POST API để lưu thông tin thiết kế bánh

```
@RestController
@RequestMapping(path = "/design", produces = "application/json")
@CrossOrigin(origins = "*")
public class DesignTacoController {
    private TacoRepository tacoRepo;

    @Autowired
    EntityLinks entityLinks;

    public DesignTacoController(TacoRepository tacoRepo) {
        this.tacoRepo = tacoRepo;
    }

    @GetMapping("/recent")
    public Iterable<Taco> recentTacos() {
        return tacoRepo.findAll();
    }
}
```

# Tạo POST API để lưu thông tin thiết kế bánh

```
@GetMapping("/{id}")
public Taco tacoById(@PathVariable("id") Long id) {
    Optional<Taco> optTaco = tacoRepo.findById(id);
    if (optTaco.isPresent()) {
        return optTaco.get();
    }
    return null;
}

@PostMapping(consumes = "application/json")
@ResponseStatus(HttpStatus.CREATED)
public Taco postTaco(@RequestBody Taco taco) {
    return tacoRepo.save(taco);
}
}
```

## Tạo POST API để lưu thông tin thiết kế bánh

- Ngoài các chú giải cho Rest Controller và các phương thức GET như Ingredient Controller, cần bổ sung thêm phương thức xử lý POST request.
- Phương thức này được đánh dấu bằng `@PostMapping`, bổ sung thêm thuộc tính `consumes="application/json"` cho biết phương thức có thể xử lý dữ liệu dạng JSON.
- Tham số taco được đánh dấu bằng `@RequestBody` để cho biết thân của request cần được chuyển đổi thành đối tượng Taco và gắn vào thành tham số.
- Phương thức thực hiện nhiệm vụ đơn giản là lưu thông tin đối tượng Taco vào CSDL qua Taco Repository.

## Sử dụng POST API vừa tạo

- Quay lại Project Client
- Tiếp tục sử dụng Rest Template để gửi dữ liệu tới POST API để xử lý lưu dữ liệu vào CSDL
- Sử dụng phương thức `postForObject(url, object, class)` để gọi API:
  - url: đường dẫn của API
  - object: đối tượng sẽ được gửi kèm POST request tới API
  - class: lớp của đối tượng
  - Ví dụ:

```
rest.postForObject("http://localhost:8080/ingredients", ingredient, Ingredient.class);
```

# Chỉnh sửa phương thức processDesign()

@PostMapping

```
public String processDesign(@RequestParam("ingredients") String
    ingredientIds, @RequestParam("name") String name) {
    List<Ingredient> ingredients = new ArrayList<Ingredient>();
    for (String ingredientId : ingredientIds.split(",")) {
        Ingredient ingredient = rest.getForObject("http://localhost:8080/
            ingredients/{id}", Ingredient.class, ingredientId);
        ingredients.add(ingredient);
    }
    Taco taco = new Taco();
    taco.setName(name);
    taco.setIngredients(ingredients);
    System.out.println(taco);
    rest.postForObject("http://localhost:8080/design", taco, Taco.class);
    return "redirect:/orders/current";
}
```



## Chỉnh sửa phương thức `processDesign()`

- Thông thường, form trong Thymeleaf sẽ tự động gán các trường của form vào đối tượng và truyền tới controller.  
VD phương thức `processDesign()` ở lớp `DesignTacoController` có tham số đối tượng `taco` được truyền từ form.
- Tuy nhiên, các trường được gán tự động bởi Thymeleaf luôn có kiểu `String`, trong khi đối tượng `Taco` có thuộc tính `ingredients` là 1 danh sách các đối tượng `Ingredient` (phục vụ cho JPA). Do vậy cần phải trích xuất các `Request Param` và tạo đối tượng 1 cách thủ công.

## Chỉnh sửa phương thức `processDesign()`

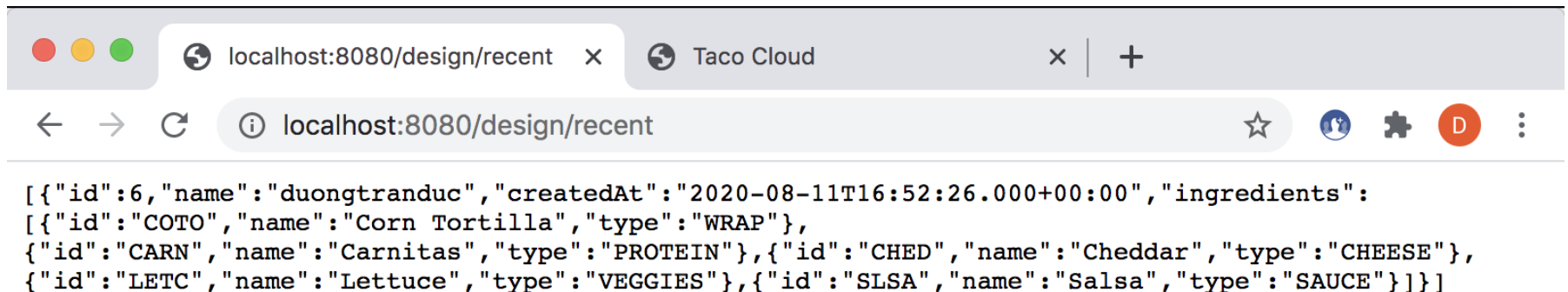
- Các request param sẽ được truyền vào phương thức dạng `@RequestParam(param_name) Type var_name)`.

VD: `@RequestParam("ingredients") String ingredientIds`

- Phương thức `processDesign()` sẽ lấy danh sách các mã Ingredient được chọn từ form, gọi API `getForObject()` để lấy về đối tượng Ingredient tương ứng và add vào một list các Ingredient.
- Một đối tượng Taco sẽ được tạo ra từ param name và ingredients.
- Đối tượng Taco được tạo ra sẽ dùng cho `postForObject()` để gửi tới POST API nhằm thêm vào CSDL.

## Chỉnh sửa phương thức processDesign()

- Sau khi tiến hành các chỉnh sửa, chạy ứng dụng tại localhost:8081/design để chọn thành phần và tạo thiết kế bánh.
- Gọi API localhost:8080/design/recent để kiểm tra kết quả là thiết kế bánh đã được thêm vào CSDL.



```
[{"id":6,"name":"duongtranduc","createdAt":"2020-08-11T16:52:26.000+00:00","ingredients": [{"id":"COTO","name":"Corn Tortilla","type":"WRAP"}, {"id":"CARN","name":"Carnitas","type":"PROTEIN"}, {"id":"CHED","name":"Cheddar","type":"CHEESE"}, {"id":"LETC","name":"Lettuce","type":"VEGGIES"}, {"id":"SLSA","name":"Salsa","type":"SAUCE"}]}
```

## Tiếp tục ...

- Tạo các API cho OrderController tương tự DesignTacoController
- Chỉnh sửa project Client để thêm order mới vào CSDL tương tự việc thêm Taco Design vào CSDL
- Tạo thêm các API POST, PUT, DELETE cho IngredientController và thực hiện các thao tác thêm/xoá/sửa từ phía Client.
- Lưu ý: Đối với các API cho đối tượng Ingredient, do lớp này không có liên kết với lớp khác nên có thể lấy trực tiếp đối tượng được chuyển về từ Thymeleaf mà không cần phải trích xuất thủ công từ Request Param.

## Tiếp tục ...

- Phương thức post:

- Server:

```
@PostMapping(consumes="application/json")
@ResponseStatus(HttpStatus.CREATED)
public Ingredient postIngredient(@RequestBody Ingredient
ingredient) {
    return ingredientRepo.save(ingredient);
}
```

- Client:

```
rest.postForObject("http://localhost:8080/ingredients",
ingredient, Ingredient.class);
```

## Tiếp tục ...

- Phương thức put:

- Server:

```
@PutMapping("/{ingredientId}")  
  
public Order putIngredient(@RequestBody Ingredient  
ingredient) {  
    return ingredientRepo.save(ingredient);  
}
```

- Client:

```
rest.put("http://localhost:8080/ingredients/{id}",  
ingredient, ingredient.getId());
```

## Tiếp tục ...

- Phương thức delete:

- Server:

```
@DeleteMapping("/{ingredientId}")  
  
public void deleteIngredient(@PathVariable("ingredientId")  
Long ingredientId) {  
    try {  
        ingredientRepo.deleteById(ingredientId);  
    } catch (EmptyResultDataAccessException e) {}  
}
```

- Client:

```
rest.delete("http://localhost:8080/ingredients/{id}",  
ingredient.getId()):
```