Representational State Transfer (REST)

REST

- **Re**presentational **S**tate **T**ransfer
- Not a specific protocol or technology. Rather an architectural style
- Back to the principles of the web
  - Use URLs to identify Resources
  - Four HTTP commands let you operate uniformly on resources:
    - **POST** (Create)
    - **GET** (Read)
    - **PUT** (Update)
    - **DELETE** (Delete)

REST vs. RPC and SOAP-based Web Services

- RPC: proliferation of vocabulary

```
getUser(), addUser(), removeUser(), updateUser(),
getLocation(), addLocation(), removeLocation(), ...

service = new WebService("example.com");
service.getUser(id);
service.removeLocation(id);
```

- REST: reuse of vocabulary

```
http://example.com/users
http://example.com/users/{user}
http://example.com/locations

user = new Resource("example.com/users/001");
user.get();
location = new Resource("example.com/locations/a");
location.delete();
```

RESTful architectures

- Minimal, uniform interface to manipulate resources
- Also:
  - Stateless interactions between clients and servers
  - Caching: responses indicate whether they can be cached or not
  - Layered system: clients cannot tell whether they talk directly to the server or to a proxy server
- HTTP is just one example of a RESTful design. REST is a more general design principle
- JSON messages used in most cases

Resource-oriented services

- REST works well with resource-oriented services
- Amazon S3 storage service
- Document stores: Apache Couch DB, Elasticsearch, …
- OpenAI API to interact with AI Models (e.g. GPT, DALL-E, …)
- Atom Publishing Protocol
- Twitter (X)
- …

Example: DALL-E RESTful API

```
#generations
curl https://api.openai.com/v1/images/generations \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -d '{ "prompt": "a photo of a happy corgi puppy sitting and facing forward, studio light, longshot",
      "n":1,
      "size":"1024x1024"
   }'
```

https://openai.com/blog/dall-e-api-now-available-in-public-beta

Example: DALL-E RESTful API

```
#edits
curl https://api.openai.com/v1/images/edits \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -F image="@/Users/openai/happy_corgi.png" \
  -F mask="@/Users/openai/mask.png" \
  -F prompt="a photo of a happy corgi puppy with fancy sunglasses on sitting and facing forward, studio light, longshot" \
  -F n=1 \
  -F size="1024x1024"
```

https://openai.com/blog/dall-e-api-now-available-in-public-beta

Example: DALL-E RESTful API

```
#variations
curl https://api.openai.com/v1/images/variations \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -F image="@/Users/openai/corgi_with_sunglasses.png" \
  -F n=4 \
  -F size="1024x1024"
```

https://openai.com/blog/dall-e-api-now-available-in-public-beta

Example: RESTful Twitter API

- To retrieve a user's twitter timeline:

```
HTTP GET https://api.twitter.com/1.1/statuses/user_timeline.json?user_id=user
```

- Returns a JSON-formatted document containing a list of tweets:

```
[
 {
    "id": 240859602684612608,
    "text": "A tweet",
    "retweet_count": 121,
    "created_at": "Wed Aug 29 17:12:58 +0000 2012",
    "favorited": false,
    ...
 },
 {
  ...
 }
]
```

Example: RESTful Twitter API

- Can delve deeper and retrieve details of a tweet:

```
HTTP GET https://api.twitter.com/1.1/statuses/show.json?id=240859602684612608
```

- Returns a JSON-formatted document containing a single tweet:


```
{
  "id": 240859602684612608,
  "text": "A tweet",
  "retweet_count": 121,
  "created_at": "Wed Aug 29 17:12:58 +0000 2012",
  "favorited": false,
  "entities": {
    "urls": [...],
    "hashtags": [...],
    "user_mentions": [...]
  },
  ...
}
```

Example: RESTful Twitter API

- Can also post new tweets:

```
HTTP POST https://api.twitter.com/1.1/statuses/update.json
status=Hello%20%23world
```

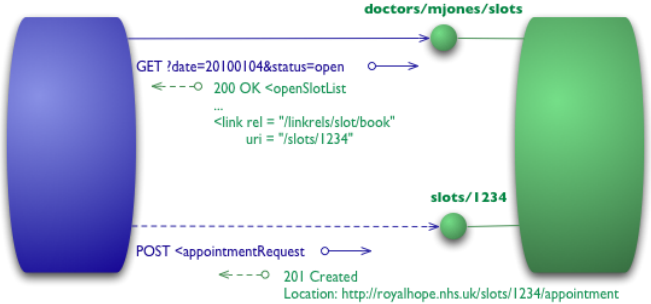
- Returns a JSON-formatted document describing the posted tweet:

```
{
  "id": 243145735212777472,
  "text": "Hello #world",
  "retweet_count": 0,
  "created_at": "Wed Aug 29 17:37:58 +0000 2012",
  "favorited": false,
  ...
}
```

Richardson Maturity Model for REST

Common Lab

- Level 0 : HTTP only used as transport system, only 1 endpoint
  - E.g. SOAP
- Level 1 – Resources : Uses different URIs, but with a fixed HTTP verb
- Level 2 – Verbs : Different HTTP verbs are used per URI
- Level 3 – Hypermedia controls : Links included in the response for follow-up actions



https://martinfowler.com/articles/richardsonMaturityModel.html

KU LEUVEN DistriNet

Lab: Food Delivery Application

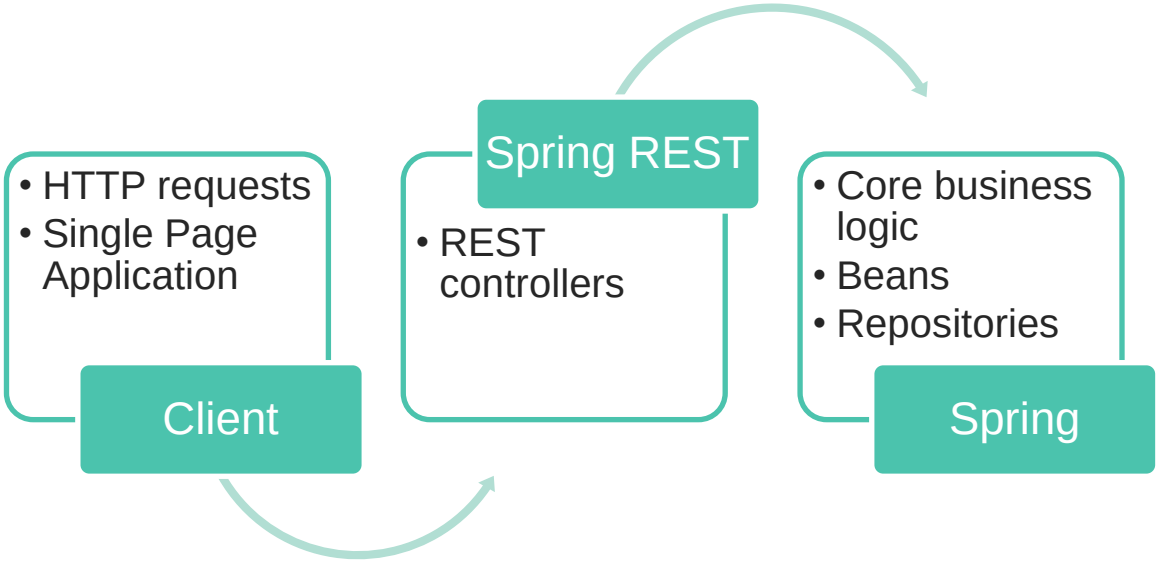
- Online service of a food delivery company
  - Inspect the menu
  - Order Meals
- 2 Parts:
  - REST Lab: Code-First
  - OpenAPI Lab: API-First

REST Lab

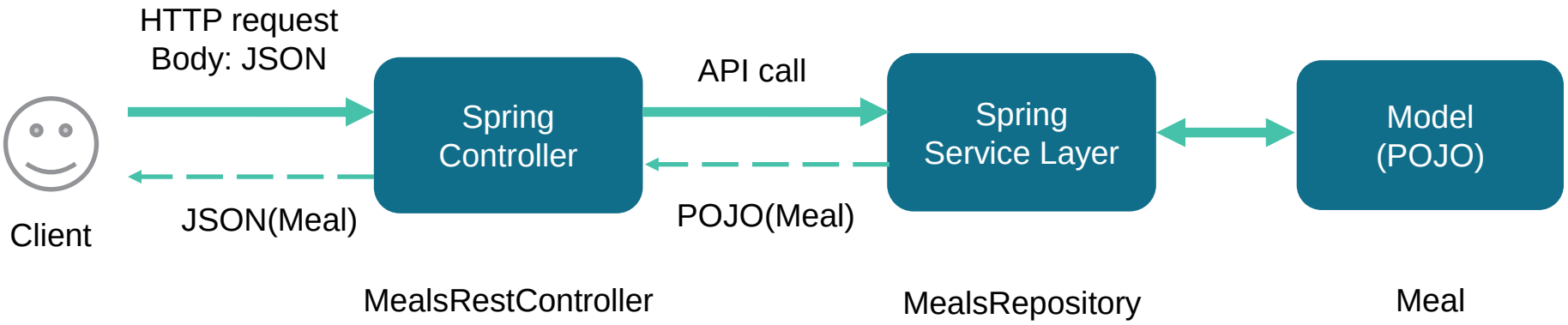
- Goal
  - RESTful Services in Java
  - Code-First approach
  - Spring Framework
  - Level 2 & Level 3 Services (Richardson Maturity Model)
- Approach
  - Basic Java Project provided
  - Test and Extend both controllers

REST Lab: Architecture

› Spring REST services



REST Lab: Architecture



REST LAB: Domain Entities

› Plain Old Java Objects

```
public class Meal {

    protected String id;
    protected String name;
    protected Integer kcal;
    protected Double price;
    protected String description;
    protected MealType mealType;

    …
}
```

```
public enum MealType {

    VEGAN("vegan"),
    VEGGIE("veggie"),
    MEAT("meat"),
    FISH("fish");

    …
}
```

REST LAB: Repository

```
@Component
public class MealsRepository {
   // map: id -> meal
   private static final Map<String, Meal> meals = new HashMap<>();

   @PostConstruct
   public void initData() {

      Meal a = new Meal();
      a.setId("5268203c-de76-4921-a3e3-439db69c462a");
      a.setName("Steak");
      a.setDescription("Steak with fries");
      a.setMealType(MealType.MEAT);
      a.setKcal(1100);
      a.setPrice((10.00));

      meals.put(a.getId(), a);

      …
   }

… // Operations
}
```

Executed after Constructor

```
public Optional<Meal> findMeal(String id) {
      Assert.notNull(id, "The meal id must not be null");
      Meal meal = meals.get(id);
      return Optional.ofNullable(meal);
   }

   public Collection<Meal> getAllMeal() {
      return meals.values();
   }
```

REST LAB: Spring REST Controller

```
@RestController
public class MealsRestRpcStyleController {

   private final MealsRepository mealsRepository;

   @Autowired
   MealsRestRpcStyleController(MealsRepository mealsRepository) {
      this.mealsRepository = mealsRepository;
   }

   @GetMapping("/restrpc/meals/{id}")
   Meal getMealById(@PathVariable String id) {
      Optional<Meal> meal = mealsRepository.findMeal(id);

      return meal.orElseThrow(() -> new MealNotFoundException(id));
   }

   @GetMapping("/restrpc/meals")
   Collection<Meal> getMeals() { return mealsRepository.getAllMeal();}
}
```

Allows auto-detection in Spring Framework

Defines URL of operation

@Pathvariable
@RequestBody
@RequestParam

Demo - RESTful Services

OpenAPI Specification (OAS)

OpenAPI

- https://www.openapis.org/
- Previously Swagger Specification
- Standard way to describe REST APIs
- Language Agnostic
- Human- and Machine-readable interface definition language (IDL)
- Documentation, code generation, testing, configuration (import in tools and appliances)

OpenAPI Specification Example

```yaml
openapi: 3.0.3
info:
  title: Resto
  description: Delicious Meal API
  version: v1.0.0
externalDocs:
  description: Find out more about Swagger
  url: https://swagger.io
servers:
  - url: http://localhost:8080
    description: Our Local Server
tags:
  - description: Everything about our delightful Resto Services
    name: Resto Service
```

```yaml
paths:
  /meals:
    get:
      tags:
        - meals
      summary: Retrieve all meals
      description: Find all meals
      operationId: getMeals
      responses:
        "200":
          description: OK
          content:
            application/json:
              schema:
                items:
                  $ref: '#/components/schemas/Meal'
        "404":
          description: No Meals found
```

https://swagger.io/specification/

OpenAPI Specification Example

```yaml
components:
  schemas:
    Meal:
      type: object
      properties:
        id:
          type: string
          description: Unique id of the meal
          format: uuid
        mealType:
          type: string
          description: The type of meal
          enum:
            - VEGAN
            - VEGGIE
            - MEAT
            - FISH
      required:
        - name
        - mealType
      description: A Delicious meal
```

```yaml
  responses:
    UnauthorizedError:
      description: Authentication information is missing or invalid
      headers:
        WWW_Authenticate:
          schema:
            type: string
```

```yaml
  securitySchemes:
    basicAuth:
      type: http
      scheme: basic
    apiKey:
      type: apiKey
      in: header        # can be "header", "query" or "cookie"
      name: X-API-KEY
```

OpenAPI Code Generation

- OpenAPI can be used to generate
  - Client Code
  - Server Code
  - Unit Tests, Performance Tests
  - Documentation
- Many tools exist
  - Openapi-generator (OpenAPITools)
  - Swagger Codegen (SmartBear)

Lab

Swagger UI

- Visual documentation of the API
- Allows developers and customers that use the API to:
  - Visualize the API
  - Interact with the API
- Based on OpenAPI Spec
- Based on Annotation in Java Spring with Springdoc

Swagger UI

Swagger
Supported by SMARTBEAR

Resto v1.0.0 OAS3

Delicious Meal API

Find out more about Swagger

Servers

http://localhost:8080 - Our Local Server

Resto Service Everything about our delightful Resto Services

meals

GET	/meals	Retrieve all meals	getMeals
POST	/meals	Add a new meal	addMeal
GET	/meals/{id}	Get a meal by its id	getMealById
PUT	/meals/{id}	Update existing meal	updateMeal
DELETE	/meals/{id}	Remove a meal	deleteMeal

KU LEUVEN DistriNet

OpenAPI Lab

- Goal
  - RESTful Services in Java
  - API-First approach
  - Code generation from an OpenAPI specification
- Approach
  - Basic Java Project provided
  - Generate code & Test
  - Extend the OpenAPI specification and the code

OpenAPI Lab: Annotations

- Annotations for dynamic OpenAPI generation

```
@Operation(summary = "Get a meal by its id", description = "Get a meal by id description")
@ApiResponses(value = {
    @ApiResponse(responseCode = "200", description = "Found the meal",
        content = {@Content(mediaType = "application/json", schema = @Schema(implementation = Meal.class))}),
    @ApiResponse(responseCode = "404", description = "Meal not found", content = @Content)})
@GetMapping("/rest/meals/{id}")
ResponseEntity<?> getMealById(
    @Parameter(description = "Id of the meal", schema = @Schema(format = "uuid", type = "string")) @PathVariable String id) {
…
}
```

- http://localhost:8080/v3/api-docs
- http://localhost:8080/swagger-ui/index.html

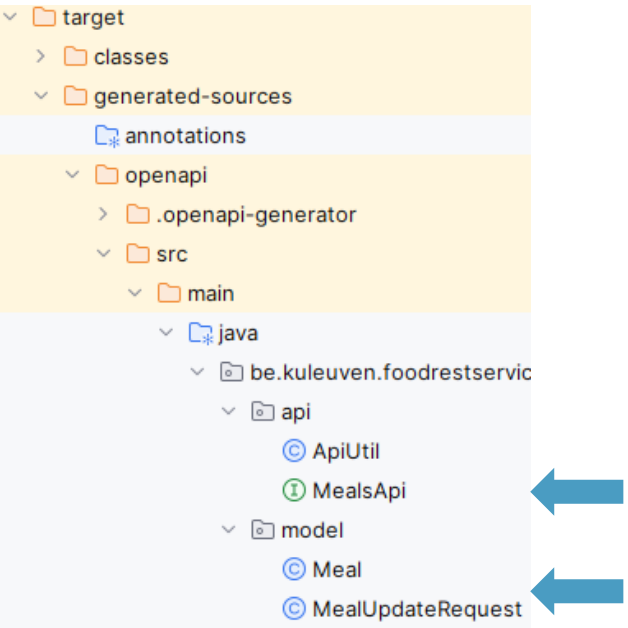
OpenAPI Lab: OpenAPI-Generator

Lab

- Interface Generation
  - No modification to generated files
  - Change OpenAPI Spec and regeneration OK
  - Few files in source control (OpenAPI spec + own classes)
- Full Generation
  - All classes, controllers, … are generated
  - Can be modified and tweaked at will
  - Regeneration overwrites generated files!

OpenAPI Lab: OpenAPI-Generator

- API Interface Generated
- POJO objects generated
- Controller implements API Interface



OpenAPI Lab: Domain Entities

```yaml
components:
  schemas:
    Meal:
      type: object
      properties:
        id:
          type: string
          description: Unique id of the meal
          format: uuid
        mealType:
          type: string
          description: The type of meal
          enum:
            - VEGAN
            - VEGGIE
            - MEAT
            - FISH
      required:
        - name
        - mealType
      description: A Delicious meal
```

```java
@Schema(name = "Meal", description = "A Delicious meal")
@Generated(value = "...", date = ".")
public class Meal {

 private UUID id;

 ...

 /**
  * The type of meal
  */
 public enum MealTypeEnum {
  VEGAN("VEGAN"),
  ...
 }
...
}
```

OpenAPI Lab: API Interface

```
public interface MealsApi {

   default Optional<NativeWebRequest> getRequest() {
      return Optional.empty();
   }

   …
   @RequestMapping(
      method = RequestMethod.POST,
      value = "/meals",
      produces = { "application/json" },
      consumes = { "application/json" }
   )
   default ResponseEntity<Object> addMeal(
      @Parameter(name = "MealUpdateRequest", description = "", required = true) @Valid
@RequestBody MealUpdateRequest mealUpdateRequest
   ) {
      return new ResponseEntity<>(HttpStatus.NOT_IMPLEMENTED);

   }
```

URL Path defined in interface

All operations exist, can be implemented one at a time

OpenAPI Lab: Repository

- Similar to Repository from RESTful project
- Refers to generated domain entities

OpenAPI Lab: Controller

```
@RestController
public class MealsController implements MealsApi {

   private static final MealsRepository mealsRepository = new MealsRepository();

   @Override
   public ResponseEntity<List<Meal>> getMeals() {
      return ResponseEntity.ok(mealsRepository.getAllMeals());
   }

   @Override
   public ResponseEntity<Object> addMeal(MealUpdateRequest mealUpdateRequest) {
      Meal newMeal = mealsRepository.addMeal(mealUpdateRequest);

      return ResponseEntity.created(ServletUriComponentsBuilder.fromCurrentRequest().path(newMeal.getId().toString()).build().toUri()).body(newMeal);
   }
...
}
```

Demo – OpenAPI-Generator