MicroProfile OpenAPI

Code First or Design First?

Peter Steiner – 2019-10-24

eclipsecon Europe 2019

Peter Steiner

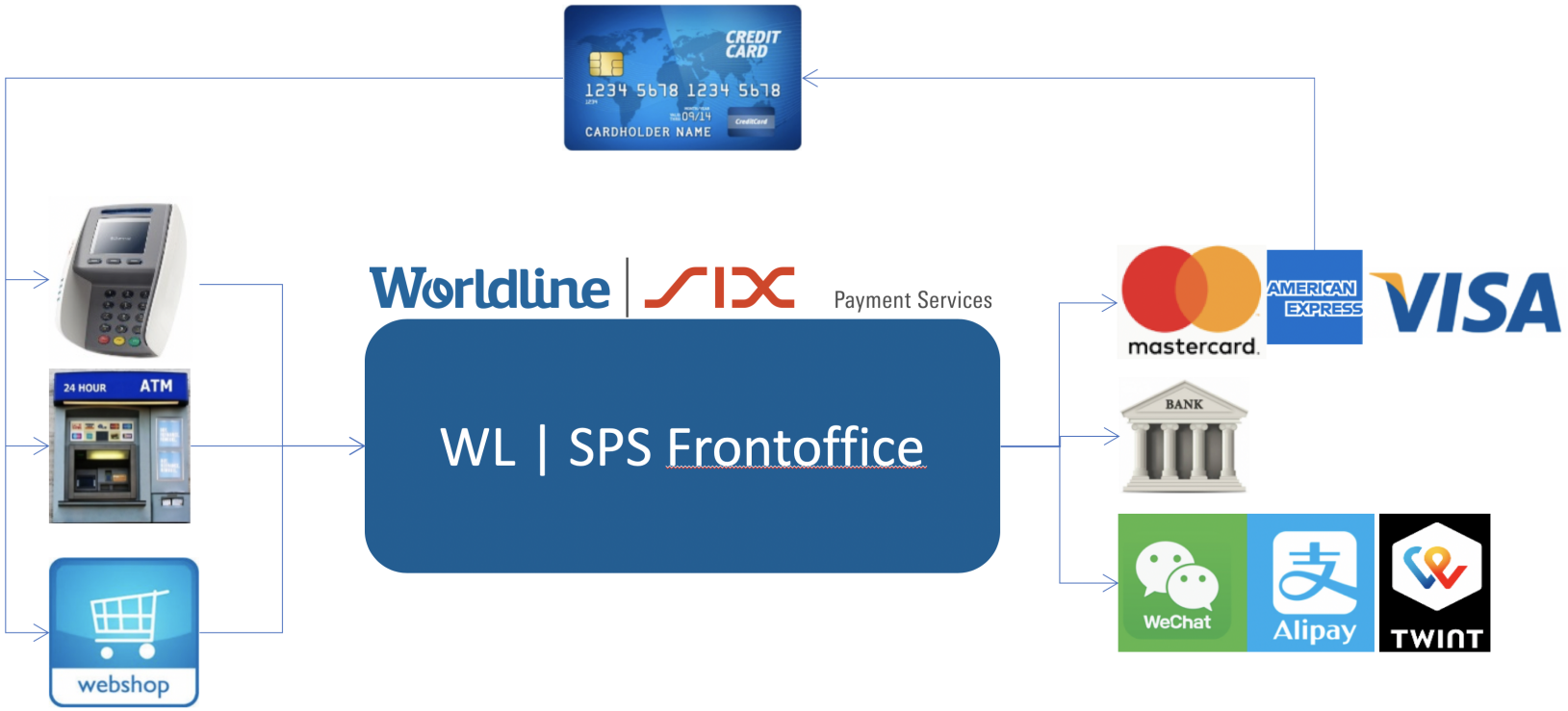
Software Architect
Java Developer

pe-st

pesche

Worldline | SIX Payment Services

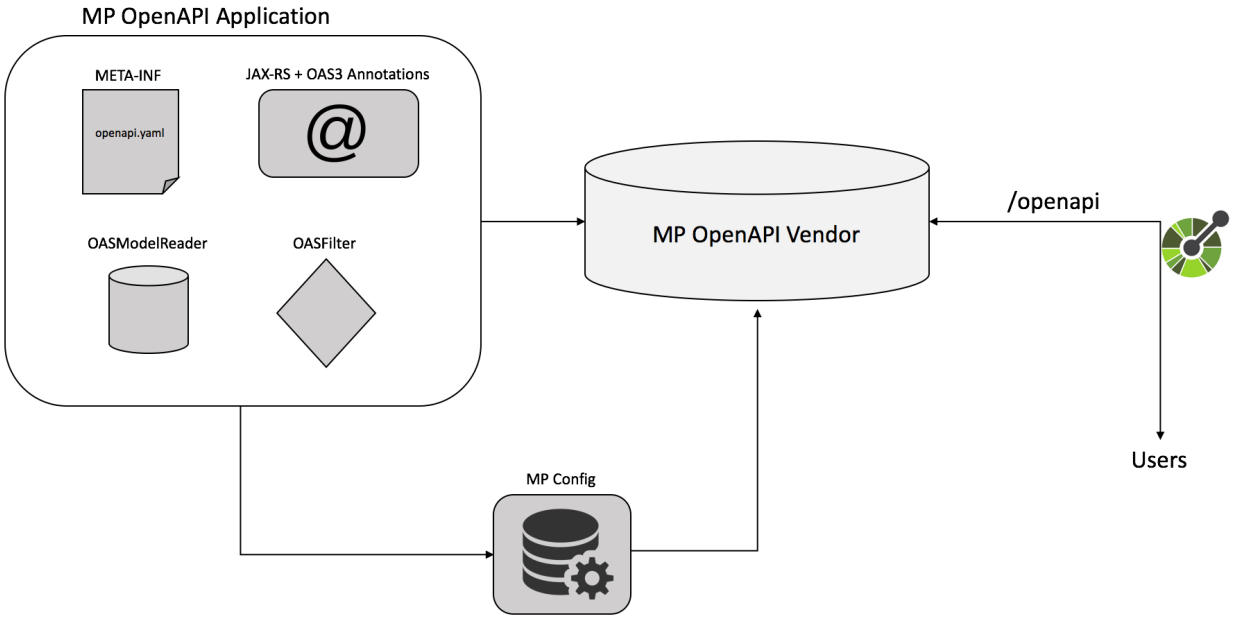
Worldline | SIX Payment Services – Frontoffice



Agenda / Goal

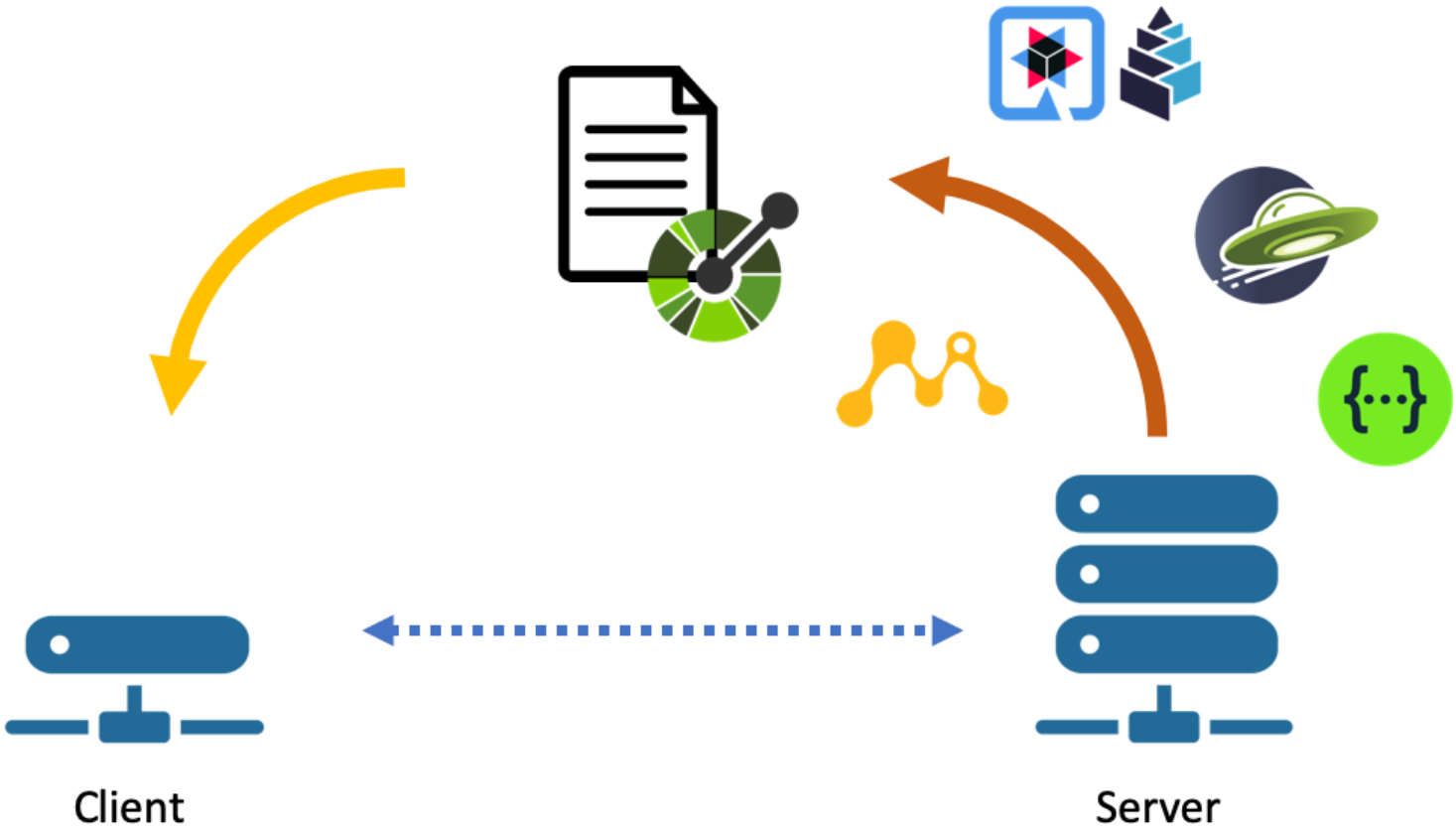
- Write a working REST service
- Have an OpenAPI Document
  - Documentation (e.g. with Swagger-UI)
  - Clients can generate interface code
- Compare Code-First and Design-First approaches

MicroProfile OpenAPI

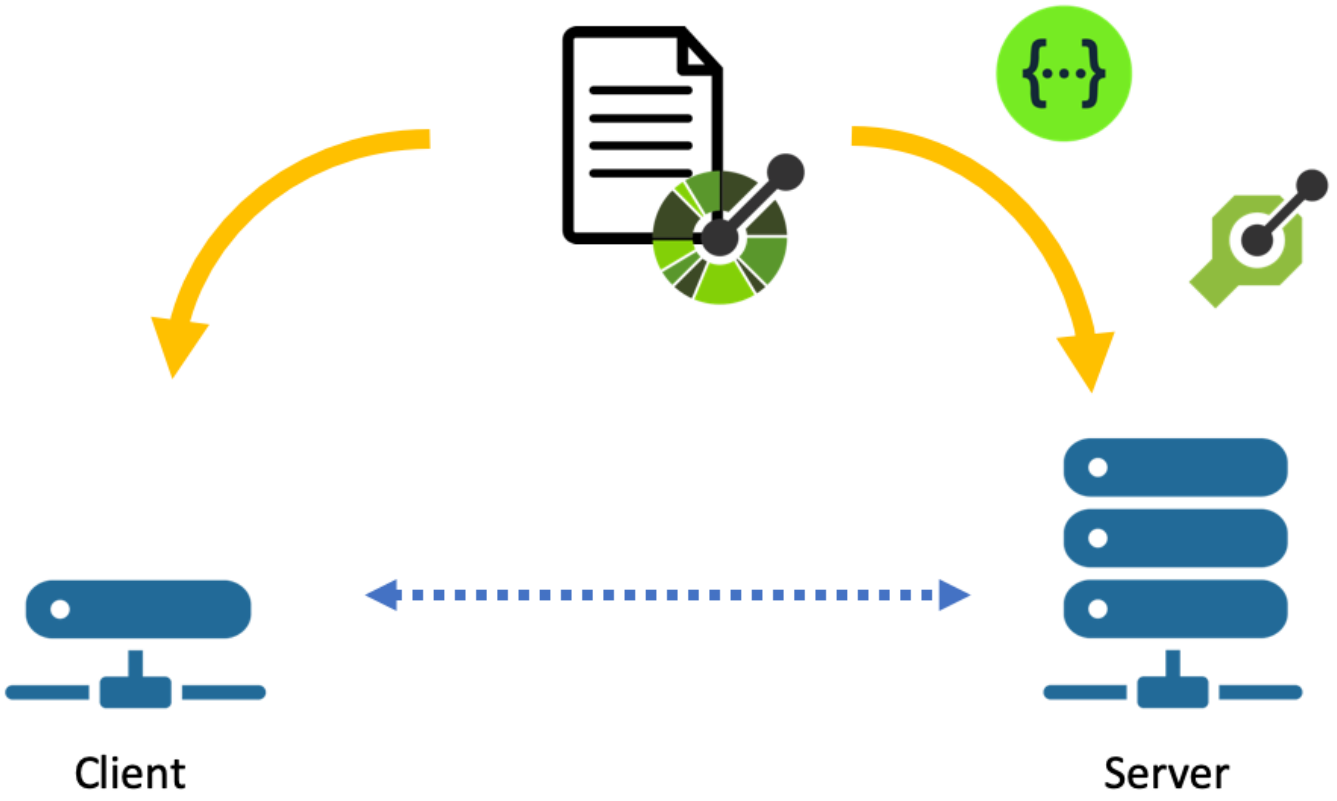


- application can provide `openapi.yaml` (or JSON) and/or annotated classes
  - JAX-RS annotations
  - MicroProfile annotations
- standard endpoint `/openapi`
- UI is not part of MicroProfile

Code First

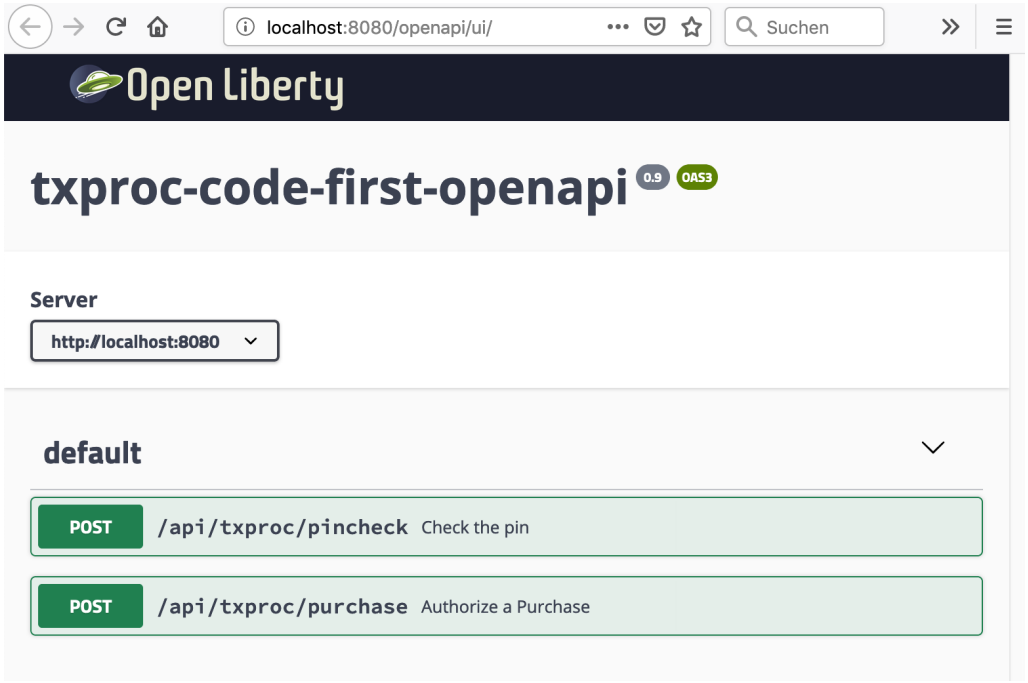


Design First

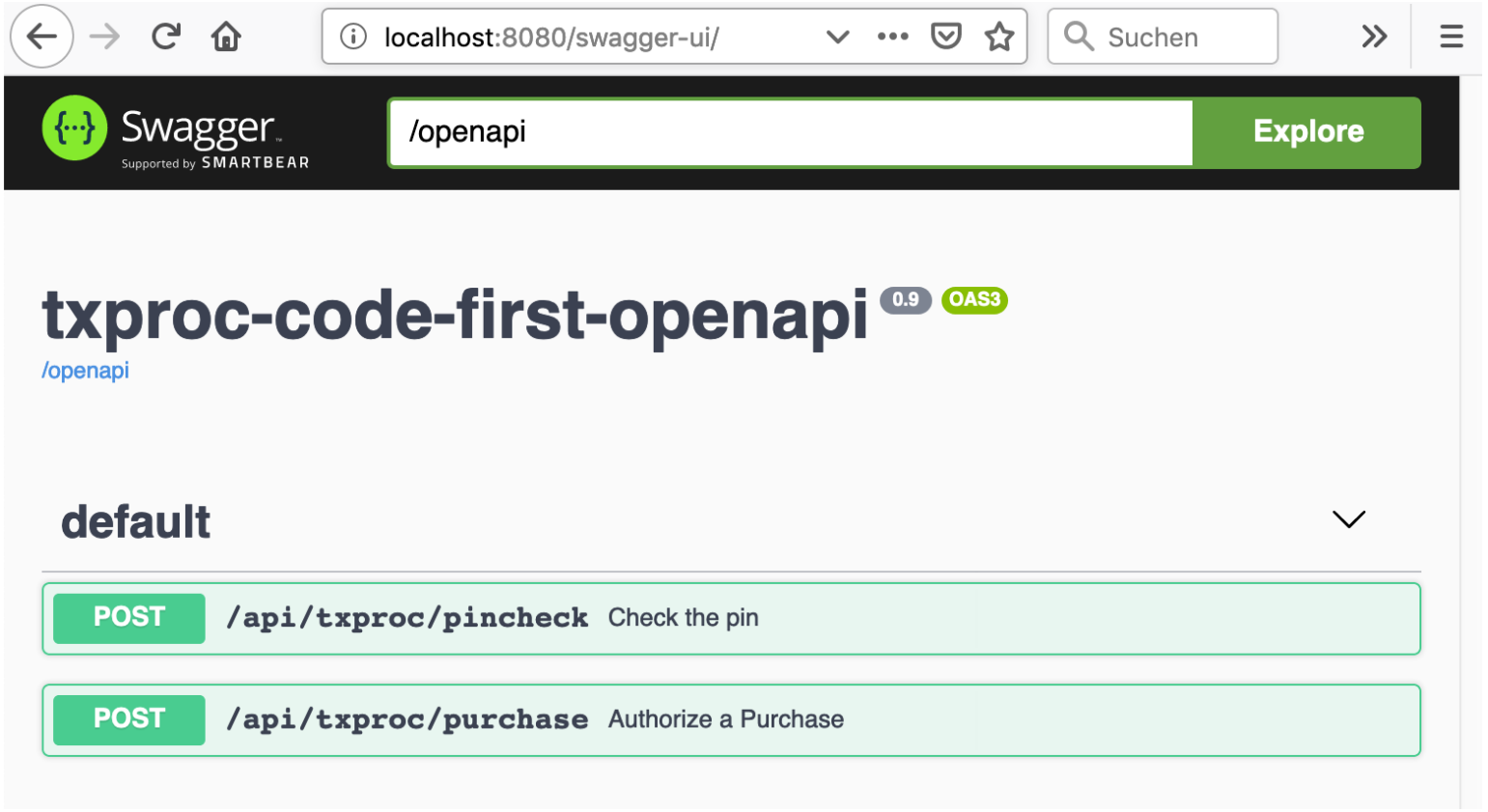


The Service "txproc"

- implemented with Quarkus and OpenLiberty (different OpenAPI scanners)
- simplified from real-world service
- multiple endpoints with shared data elements



UI with Quarkus



- Only endpoints Pincheck and Purchase, others: Reversal, Credit, Reservation etc

Swagger UI URLs

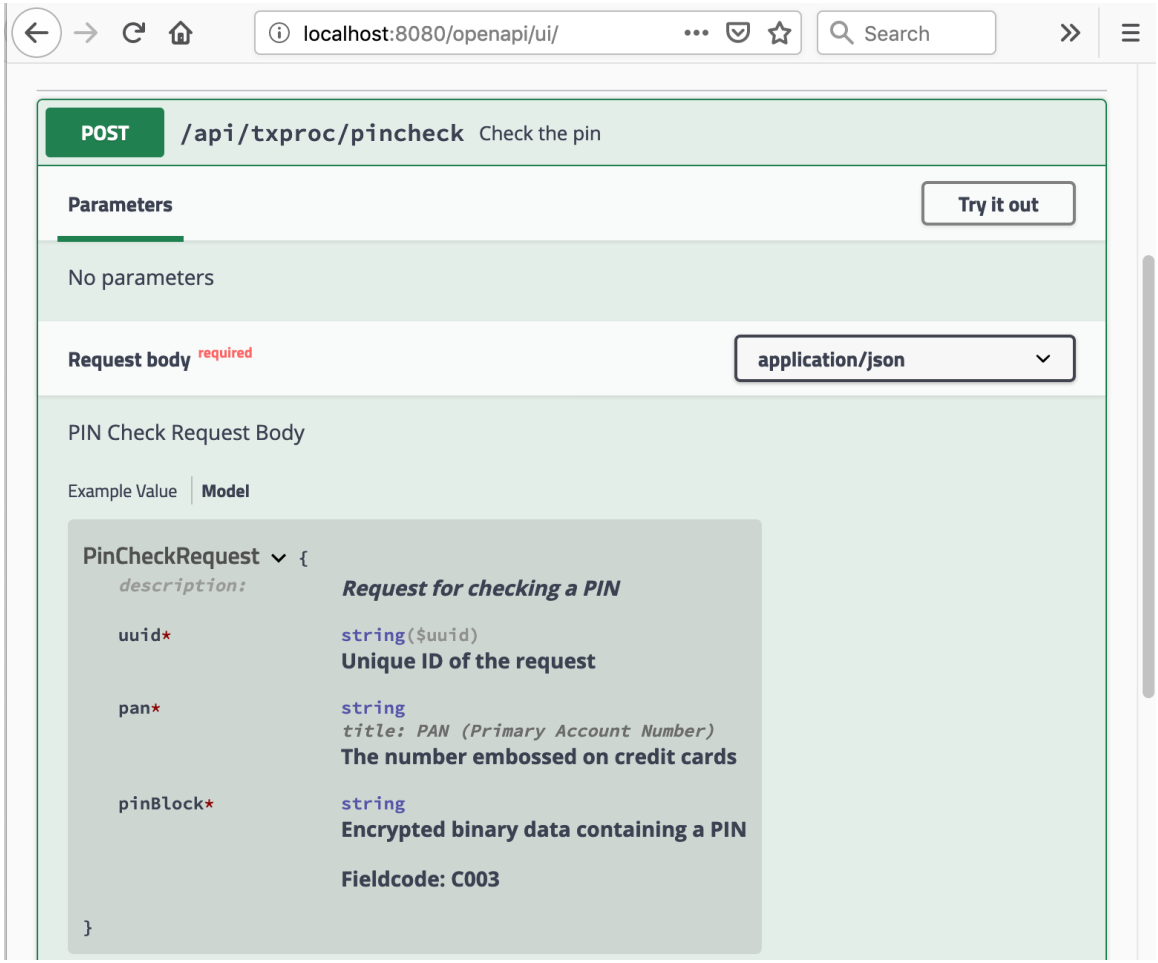
- Swagger-UI link for Quarkus: http://localhost:8080/swagger-ui
- Swagger-UI link for OpenLiberty: http://localhost:8080/openapi/ui

OpenAPI Document – First Part

```yaml
openapi: 3.0.0 (1)
info:
  title: txproc-design-first-swagger
  version: "0.9" (2)
servers:
  - url: http://localhost:8080
paths:
  /api/txproc/pincheck: (3)
    post: (4)
      summary: Check the pin
      requestBody:
        description: PIN Check Request Body
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/PinCheckRequest' (5)
```

1. OpenAPI Standard version (currently 3.0.0, 3.0.1 or 3.0.2, MP-OpenAPI links to 3.0.0)
2. API version
3. Endpoint
4. HTTP method
5. Request Body (references a schema elsewhere in the OpenAPI document)

PinCheck Request Body – UI



PinCheck Request Body – Part of OpenAPI Document

```yaml
components: (1)
  schemas:
    PinCheckRequest: (2)
      type: object
      description: Request for checking a PIN
      required: (3)
      - pan
      - pinBlock
      - uuid
      properties: (4)
        pan:
          $ref: '#/components/schemas/Pan'
        pinBlock:
          description: |- (5)
            Encrypted binary data containing a PIN

            Fieldcode: C003
          type: string
```

1. Introduces the Schema Definition part of the OpenAPI document
2. Request Body
3. List all mandatory properties (optional part, properties are optional per default)
4. Properties: inline or referencing a schema
5. Example of multiline description

Code First Actors

- Annotations
- Annotation Scan
  - MP OpenAPI Implementation Quarkus/Smallrye 
  - MP OpenAPI Implementation OpenLiberty 
  - Swagger Maven Plugin 
- Lombok
- Server to run the service and provide the `/openapi` and UI endpoints
  - Quarkus 
  - OpenLiberty 

Code First Annotation Scan – OpenLiberty

- Maven POM: Define the OpenAPI Feature via Maven dependency

```
<dependency>
    <groupId>io.openliberty.features</groupId>
    <artifactId>mpOpenAPI-1.1</artifactId>
    <type>esa</type>
    <scope>provided</scope>
</dependency>
```

- POM: You'll probably also need other features like `jaxrs-2.1`, `jsonp-1.1`, `jsonb-1.0`, `cdi-2.0` and `mpConfig-1.3`
- POM: Of course you need also the rest of the OpenLiberty incantation (feature BOM, plugin) in your `pom.xml`, see the complete example on GitHub: https://github.com/pe-st/apidocs/blob/master/code-first-openapi-openliberty/pom.xml
- Don't forget to add the features also to OpenLiberty's `server.xml`

```
<featureManager>
    <feature>jaxrs-2.1</feature>
    <feature>jsonp-1.1</feature>
    <feature>jsonb-1.0</feature>
    <feature>cdi-2.0</feature>
    <feature>mpConfig-1.3</feature>
    <feature>mpOpenAPI-1.1</feature>
</featureManager>
```

Code First Annotation Scan – Quarkus

- Define the OpenAPI Extension in your Maven POM

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-smallrye-openapi</artifactId> (1)
</dependency>
```

(1) Here you can see that Quarkus uses the SmallRye MicroProfile implementation

- Don’t forget to add also the `quarkus-resteasy-jsonb` extension; the complete POM on GitHub: https://github.com/pe-st/apidocs/blob/master/code-first-openapi-quarkus/pom.xml

Code First Annotation Scan – Swagger Plugin

- Swagger : Inspiration of MicroProfile OpenAPI
- Maven Plugin to scan the code at compile/packaging time:
  https://github.com/swagger-api/swagger-core/tree/master/modules/swagger-maven-plugin

```
<plugin>
    <groupId>io.swagger.core.v3</groupId>
    <artifactId>swagger-maven-plugin</artifactId>
    <version>2.0.9</version>
    <configuration>
        <outputFileName>openapi</outputFileName>
        <outputPath>${project.build.directory}/classes/META-INF</outputPath>
        <outputFormat>JSONANDYAML</outputFormat>
        <resourcePackages>
            <package>ch.schlau.pesche.apidocs.swagger.codefirst</package>
        </resourcePackages>
        <prettyPrint>true</prettyPrint>
    </configuration>
    <executions>...</executions>
</plugin>
```

1. in this location the file is found by MicroProfile OpenAPI implementations

Code First – Annotating (1/3)

Request body for the PIN check request

```
@Schema(description = "Request for checking a PIN") 1
@Getter 2
@Setter
public class PinCheckRequest {

    @Schema(type = "string", description = Model.UUID) 3 4
    private UUID uuid;

    // class Pan annotated with @Schema annotation 5
    private Pan pan;

    @Schema(description = Model.PIN_BLOCK)
    private String pinBlock;
}
```

1. `@Schema` : basic annotation to document classes and fields
2. using Lombok avoids the choice of annotating the field, the getter or the setter
3. without the type a struct is used instead of a flat string
4. externalizing description
5. either the field or the class itself can be annotated

Pan

```
@Schema(title = "PAN (Primary Account Number)",
        description = "The number embossed on credit cards",
        type = "string")
public class Pan {

    @Pattern(regexp = "[0-9]{12,19}")
    private String pan;

    /**
     * For data protection reasons (PCI-DSS compliance) the complete PAN must
     * not be shown unless absolutely needed. This method masks out the middle
     * digits of the PAN, allowing the result to be safely displayed
     *
     * @return masked PAN, e.g. "123456xxxxxx9012"
     */
    public String asMaskedPan() {
        // ...
    }
}
```

Code First – Annotating (2/3)

- Request body for the purchase request
- Adding requiredness for the properties

```
@Schema(description = "Request for authorizing a Purchase",
        requiredProperties = {"uuid", "pan"}) (1)
@Getter
@Setter
public class PurchaseAuthRequest {

    @Schema(type = "string", description = Model.UUID)
    private UUID uuid;

    // class Pan annotated with @Schema annotation (2)
    private Pan pan;

    // class EmvTags annotated with @Schema annotation
    private EmvTags emvTags;
}
```

(1) specify required properties on class level…

(2) …because it can’t be put here

Code First – Annotating (3/3)

(see also Design First Endpoint)

```
@Path("/pincheck") (1)
@POST (1)
@Operation(summary = "Check the pin") (2)
@APIResponse(description = "PIN Check Response", (2)
             content = @Content(schema = @Schema(implementation = PinCheckResponse.class)))
public PinCheckResponse pinCheck(
        @RequestBody(description = "PIN Check Request Body", (2)
                     content = @Content(schema = @Schema(implementation = PinCheckRequest.class))
        ) PinCheckRequest request) {

    PinCheckResponse response = new PinCheckResponse();
    // calculate response
    return response;
}
```

1. JAX-RS annotations used by the scanner
2. MicroProfile annotations (the Swagger ones are almost identical, e.g. `@ApiResponse` instead of `@APIResponse`)

Code First is tedious

Code Duplication

- Schema annotation must be copied for every endpoint…
- …unless you create a separate wrapper class for every primitive field
- …and configure the JSON framework properly
- not to speak of Javadoc or Bean Validation annotations

Code First Gotchas – Pan

- `Pan` behaves like a `String`, with additional methods
- Comes with `JsonbAdapter` to be used as follows:

```
{
  "uuid": "aaaaaaaa-bbbb-cccc-dddd-0123456789
  "pan" : "100000000042",
  "pinBlock": "magic"
}
```

Annotated as follows:

```
@Schema(title = "PAN (Primary Account Number)",
        description = "The number embossed on credit ca
        type = SchemaType.STRING) (1)
public class Pan {
    private String pan;
    // some methods
}
```

(1) tells to override the type `OBJECT` (for classes)

Resulting OpenAPI documents:

OpenLiberty + Swagger

```
properties:
  pan:
    description: The number embossed on credit card



    title: PAN (Primary Account Number)
    type: string
```

Quarkus (Smallrye)

```
properties:
  pan:
    description: The number embossed on credit card
    properties: (1)
      pan:
        type: string
    title: PAN (Primary Account Number)
    type: string
```

(1) a property of type `string` shouldn't have properties of its own

Code First Gotchas – UUID

Same issue as with `Pan`:

OpenLiberty + Swagger

```
properties:
  uuid:
    description: Unique ID of the request
    type: string
    format: uuid (1)
```

(1) only with OpenLiberty, Swagger generates no `format`

Quarkus (Smallrye)

```
properties:
  uuid:
    description: Unique ID of the request
    type: object
    properties:
      leastSigBits:
        format: int64
        type: integer
      leastSignificantBits:
        format: int64
        type: integer
      mostSigBits:
        format: int64
        type: integer
      mostSignificantBits:
        format: int64
        type: integer
```

Code First Gotchas – enum

Annotating the class instead of the field doesn't always work:

```
@Schema(description = "Tells if the PIN in the request was correct")
public class PinCheckResponse {

    @Schema(description = "Result of the request")
    public enum Code {
        OK,
        WRONG
    }

    // the enum Code already carries a @Schema annotation (1)
    private Code result;
}
```

(1) for OpenLiberty and Quarkus the annotation must be placed here

Resulting OpenAPI documents:

Swagger

```
PinCheckResponse:
  description: Tells if the PIN in the request was co
  properties:
    result:
      description: Result of the request
      enum:
      - OK
      - WRONG
      type: string
```

OpenLiberty + Quarkus (Smallrye)

```
PinCheckResponse:
  description: Tells if the PIN in the request was co
  properties:
    result:

      enum:
      - OK
      - WRONG
      type: string
```

Code First Gotchas – misc

Miscellaneous minor gotchas

- The OpenAPI documented by OpenLiberty has no deterministic order
- Quarkus doesn't create schemas for non-toplevel classes like `EmvTags` in `class PurchaseAuthRequest { EmvTags emvTags; }`

OpenLiberty + Swagger

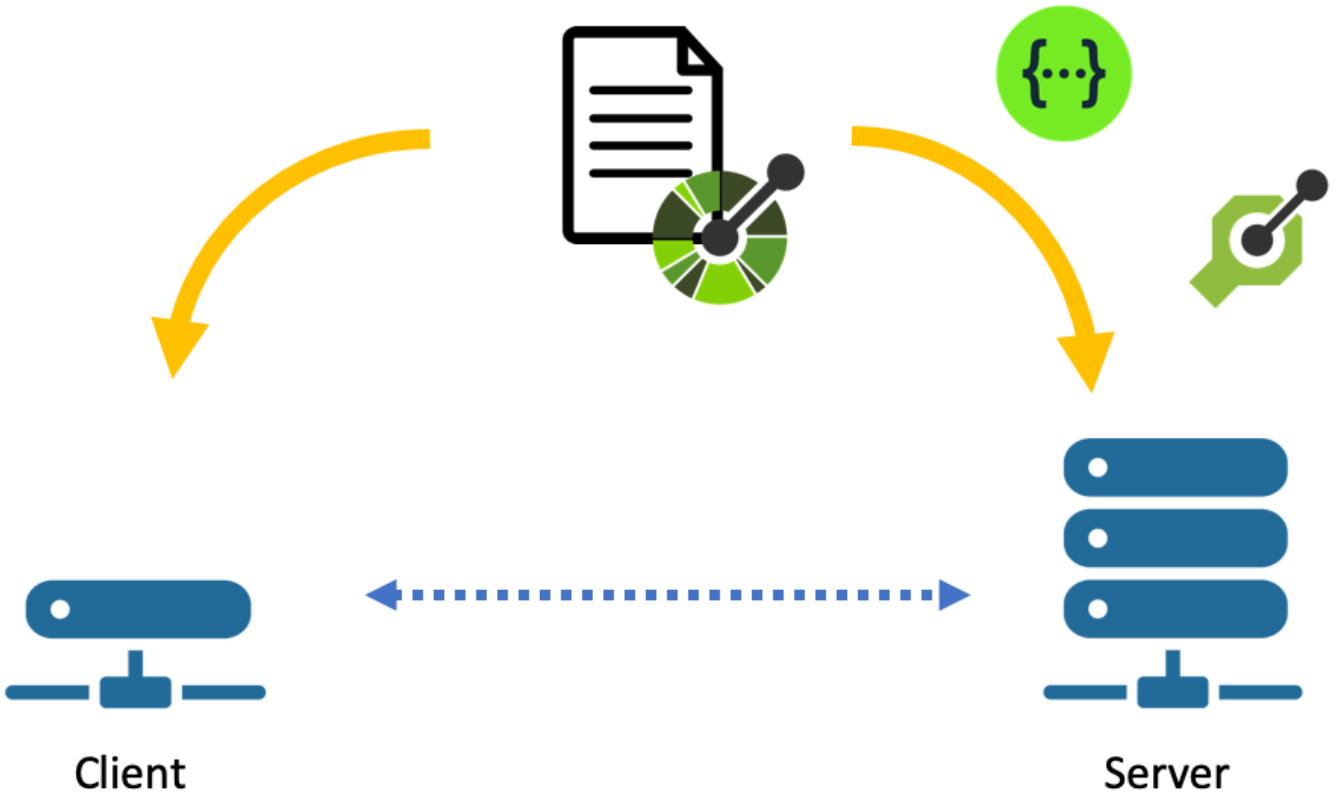
```
components:
  schemas:
    EmvTags:
      description: Collection of EMV tags
      // properties left out
      title: EmvTags
      type: object
    PurchaseAuthRequest:
      description: Request for authorizing a Purch
      properties:
        emvTags:
          $ref: '#/components/schemas/EmvTags'
```

Quarkus (Smallrye)

```
components:
  schemas:
    PurchaseAuthRequest:
      description: Request for authorizing a Purch
      properties:
        emvTags:
          description: Collection of EMV tags
          // properties left out
          title: EmvTags
          type: object
```

The proposed solution of https://github.com/smallrye/smallrye-open-api/issues/73 is to set `mp.openapi.extensions.schema-references.enable = true`, but then also other classes like `String` get referenced…

Design First



Design First Actors

- Generation with Maven Plugin for easy build integration
  - OpenAPI Generator 
  - Swagger Codegen 
- Server to run the service and provide the `/openapi` and UI endpoints
  - Quarkus 

Design First in Detail

- Both generators can generate complete server and client projects
  - including project files (`pom.xml` etc)
  - endpoint skeletons
- Scope for this talk: only the Request and Response Body POJOs (the "Models" or "Schemas") are generated
  - …using a Maven plugin
  - the endpoints are handcoded, analoguous to the code-first endpoints (better comparable, generated: ca. 12 classes, handcoded: 4 classes)
- Find a workflow where you don’t modify the generated code
  - *Generation Gap Pattern* (John Vlissides), e.g. openapi generator creates endpoints delegating to an interface (template method pattern)
  - Scope of this talk: easy, only POJOs are generated

Design First – Maven POM

```
<plugin>
    <groupId>io.swagger.codegen.v3</groupId>
    <artifactId>swagger-codegen-maven-plugin</artifactId> (1)
    <!-- ... -->
            <configuration>
                <inputSpec>${project.basedir}/src/main/resources/META-INF/openapi.yaml</inputSpec> (2)
                <language>java</language> (1)
                <generateApis>false</generateApis> (3)
                <!-- <configHelp>true</configHelp> --> (4)
                <configOptions>
                    <library>resteasy</library> (5)
                    <useBeanValidation>false</useBeanValidation>
                    <modelPackage>ch.schlau.pesche.apidocs.swagger.designfirst.generated.model</modelPacka
                </configOptions>
                <importMappings>
                    Pan=ch.schlau.pesche.apidocs.swagger.designfirst.txproc.model.Pan (6)
                </importMappings>
            </configuration>
```

1. use `org.openapitools:openapi-generator-maven-plugin` and `<generatorName>` for openapi-generator
2. this can be an URL as well
3. just generate the models, but not the endpoints
4. uncomment this line to get help output instead of doing work
5. override the default (using Gson) to generate code using Jackson
6. tell the generator that the class `Pan` already exists

Design First – Links

Complete `pom.xml` files on GitHub:

- https://github.com/pe-st/apidocs/blob/master/design-first-swagger-codegen/pom.xml
- https://github.com/pe-st/apidocs/blob/master/design-first-openapi-generator/pom.xml

Design First – Disable Scanner

- The MicroProfile annotation scanner combines per default *static OpenAPI documents* with the annotated code
- The JAX-RS annotations are mandatory
- The MicroProfile annotations aren’t needed in the generated code, but can’t currently be disabled
- The OpenAPI document is provided as source (design-first), no annotation scan needed: disable it

```
# src/main/resources/META-INF/microprofile-config.properties
mp.openapi.scan.disable = true
```

Design First Endpoint

See also the Code First Endpoint

```
import xxx.generated.model.PinCheckRequest; (1)
import xxx.generated.model.PinCheckResponse; (1)

@Path("/pincheck")
@POST
(2)
public PinCheckResponse pinCheck(PinCheckRequest request) {

    PinCheckResponse response = new PinCheckResponse();
    // calculate response
    return response;
}
```

(1) package containing a `.generated.` part to emphasize that it shouldn't be touched

(2) no OpenAPI annotations needed

Design First Gotcha – Import (1/2)

Import-Mapping (`Pan=xxx.model.Pan`), see `pom.xml`, doesn't seem to work with openapi-generator (see also https://github.com/OpenAPITools/openapi-generator/issues/3589)

```yaml
components:
  schemas:
    Pan:
      description: The number embossed on credit cards
      type: object (1)
      format: string
    PurchaseAuthRequest:
      description: Request for authorizing a Purchase
      required:
      - pan
      properties:
        pan:
          $ref: '#/components/schemas/Pan'
```

(1) type: string would make Swagger-UI prettier, but can't use imported type

Design First Gotcha – Import (2/2)

Resulting generated classes:

Swagger Codegen

```
public class PurchaseAuthRequest {

  @JsonProperty("pan")
  private xxx.model.Pan pan = null; (1)

   /**
    * Get pan
    * @return pan
   **/
  @Schema(required = true, description =
  public xxx.model.Pan getPan() { (1)
    return pan;
  }
```

(1) import-mapping `Pan` → `xxx.model.Pan` worked

OpenAPI Generator

```
public class PurchaseAuthRequest {

  public static final String JSON_PROPERTY_PAN = "pan";
  private Object pan; (1)

   /**
    * The number embossed on credit cards
    * @return pan
   **/
  @ApiModelProperty(required = true, value = "The number embossed on cre
  @JsonProperty(JSON_PROPERTY_PAN)
  @JsonInclude(value = JsonInclude.Include.ALWAYS)

  public Object getPan() { (1)
    return pan;
  }
```

(1) should be `xxx.model.Pan` instead of `Object`

Import Mapping Links

- OpenAPI Generator Import Mappings
- OpenAPI Generator Bringing your own Models

Design First or Code First – Opinions

The Swagger Blog (https://swagger.io/blog/api-design/design-first-or-code-first-api-development/) tells to choose the approach based on different procedural needs:

- Design First when the API is more important ("Mission Critical API", "Good Communication")
- Code First when "delivery speed" matters (because "automation is much easier in the code-first approach", better library support)

My point of view: Developer writing and maintaining a service with long lifecycle

- The processes are out of the picture (you might not have any influence)
- In my point of view the developer gets business requirements and shall deliver both a working service and an OpenAPI document
- This talk shows that the workflow indeed slightly differs between the two approaches

Code First Review

Pros

- No need to learn OpenAPI Documents
- Project can just be opened in the IDE (No need to generate code first)

Cons

- Portability: not every scanner creates the same `openapi.yaml`
- Portability: minor differences between Swagger and MicroProfile annotations
- Boilerplate and code duplication
- Additional integration step needed to publish the OpenAPI document to a repository

Design First Review

Pros

- One source of truth: Code, Documentation and Annotations generated from the same information
- can generate also Bean Validation annotations and Javadoc
- easy integration of an API repository (use an URL to the OpenAPI document)

Cons

- Learning Effort
- IDE integration not optimal yet

`openapi-generator` drawbacks (compared to `swagger-codegen`)

- Still uses old annotations from Swagger 1.5 in generated code (not configurable)
- Couldn't get `importMappings` to work

Take Aways

- When choosing a *code-first* approach with a SmallRye-based implementation, consider using Swagger as Scanner (with the Maven plugin at build time), the resulting OpenAPI is better
- *code-first* approach: make it a habit to check the generated OpenAPI document after code changes or scanner updates
- *design-first* approach: when you need to use existing classes in your models, Swagger Codegen is currently still better, though this might change (OpenAPI Generator gets frequently new releases)

Code Generator Comparison

swagger-codegen	openapi-generator
• Apache License 2.0 • Two different branches for OpenAPI 2.0 (master) and 3.0 (3.0.0) • Branch 3.0.0 quite different from master • Very few tests on 3.0 branch	• Apache License 2.0 • Fork from OpenAPI 2.0 branch (ca 2018-04, Fork Q&A) with added support for OpenAPI 3.0 documents • Community Driven • Import-Mappings not working (Issue 3589)

Thank You!

- Source Code: https://github.com/pe-st/apidocs
- OpenAPI Specification: https://github.com/OAI/OpenAPI-Specification/blob/3.0.2/versions/3.0.2.md
- MicroProfile OpenAPI: https://github.com/eclipse/microprofile-open-api/blob/master/spec/src/main/asciidoc/microprofile-openapi-spec.adoc

pe-st

pesche