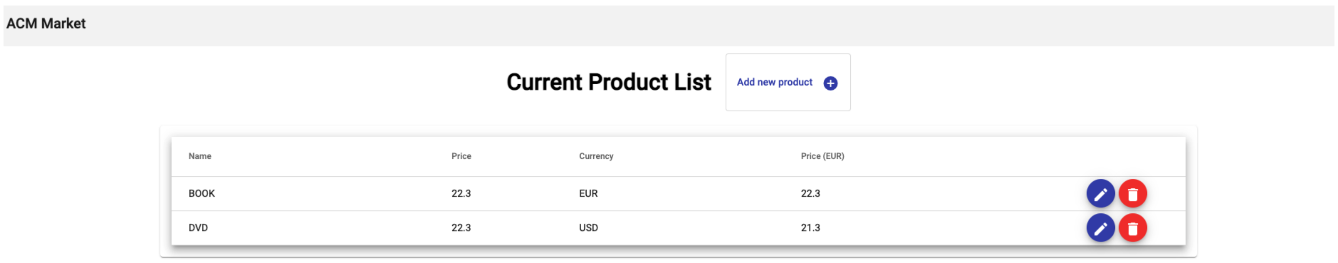
# ACM - MARKET

## Introduction

ACM-Market is a small POC, composed of 3 modules, kept on the basic Frameworks Java(Spring) and Angular without tools like RabbitMQ, Kafka or other communication Frameworks. The implementation is based on purely REST communication.

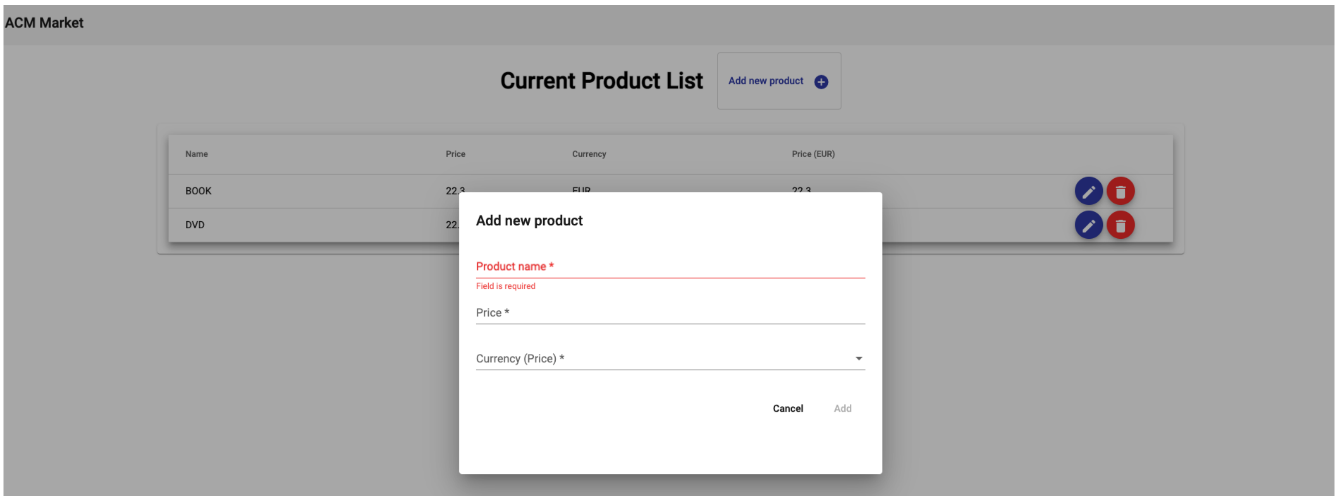
The application represent a simple CRUD interface to add/modify and delete products to the market. On adding a new product to the market, the user selects the product currency, which is transformed on the backend side to EUR, by a request to the second service in the application [exchange].

### Home screen



| Name | Price | Currency | Price (EUR) |
|---|---|---|---|
| BOOK | 22.3 | EUR | 22.3 |
| DVD | 22.3 | USD | 21.3 |

### Popup window for requesting a new product



The easiest way to start the applicat is with Docker: `docker-compose up` without docker, you need to build each module (Java/Maven, Angular/NPM) and start each module individually.

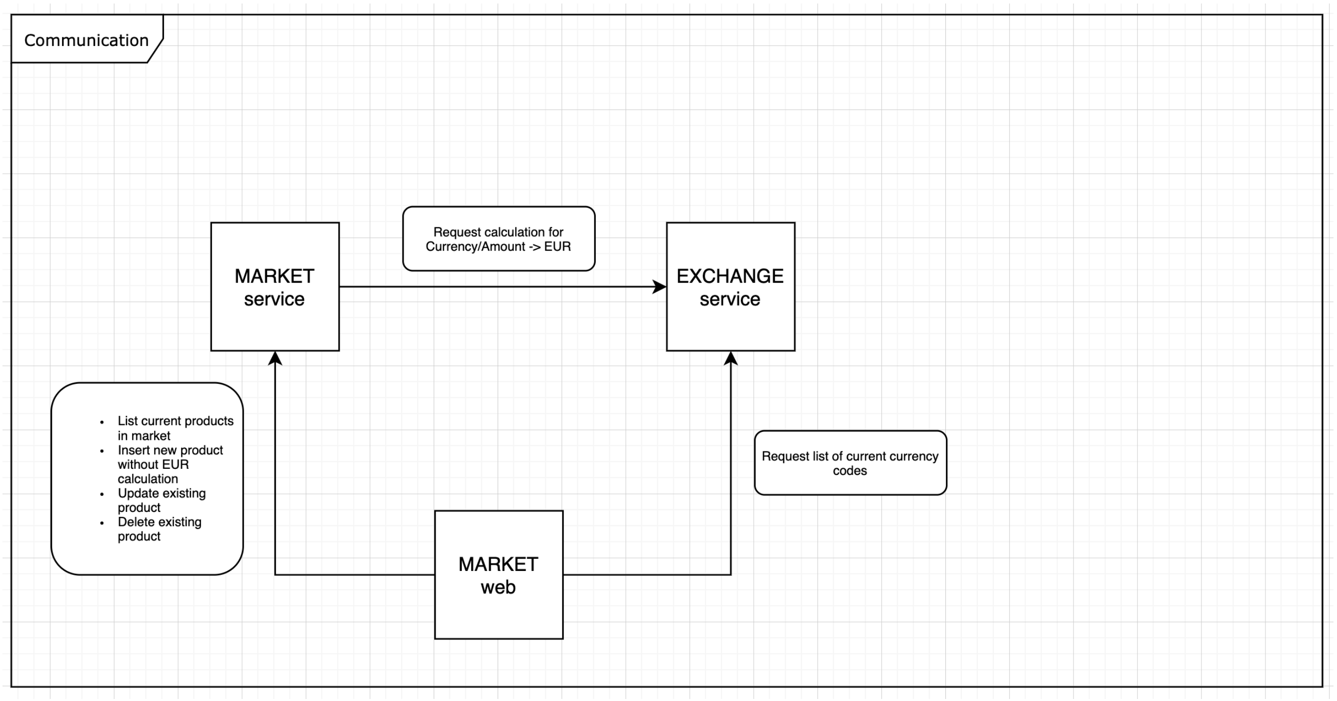
The application can be accessed:

- with Docker on: http://127.0.0.1/
- with each module individually on: http://127.0.0.1:4200/

# Component - Communication

## Schema and description

The POC is based on 3 components with different roles:

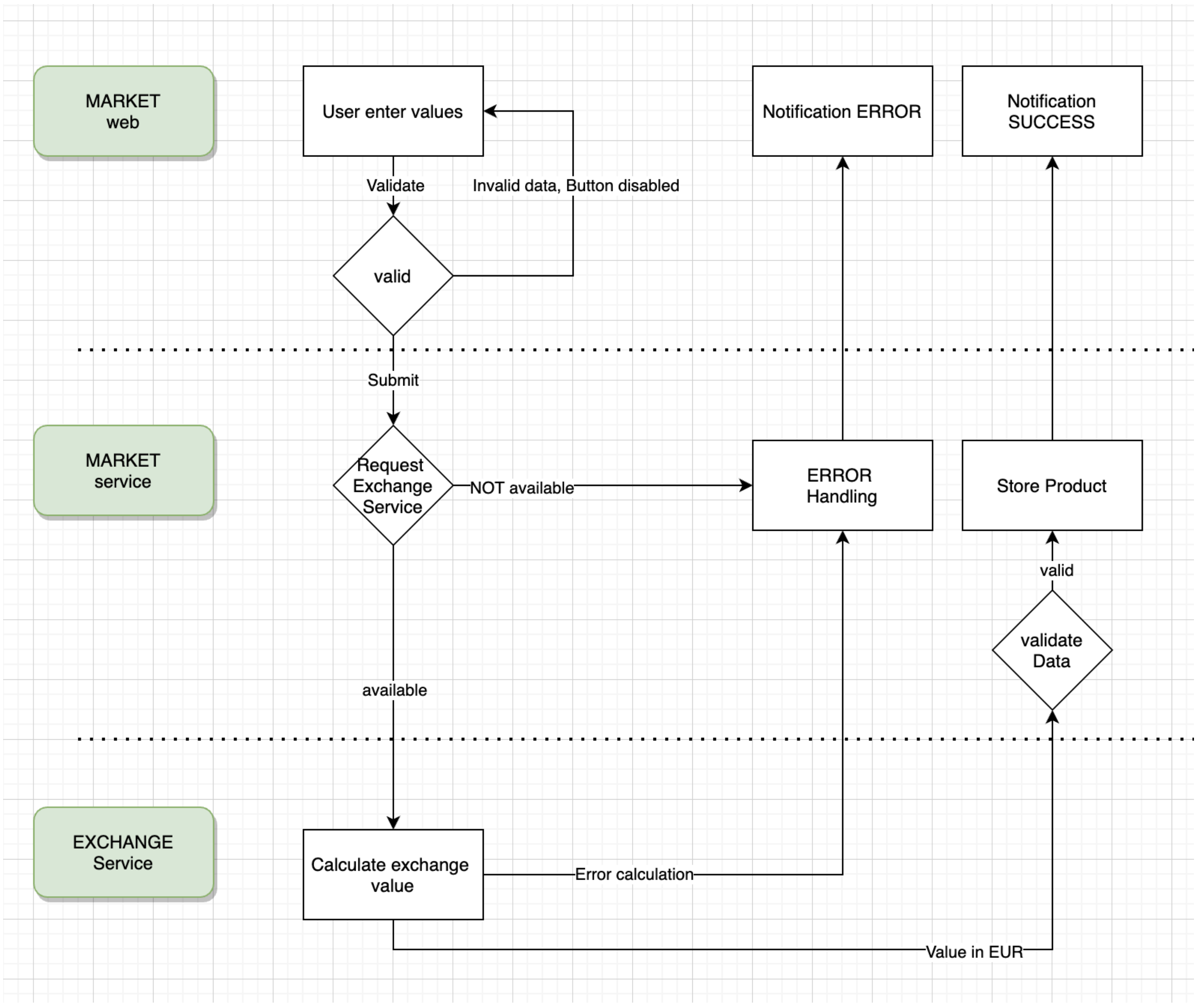

Communication
MARKET service
Request calculation for Currency/Amount -> EUR
EXCHANGE service
List current products in market
Insert new product without EUR calculation
Update existing product
Delete existing product
Request list of current currency codes
MARKET web

- EXCHANGE service
  - inventory for the latest exchange rate.
  - Calculating value in EUR by receiving the amount and 2nd currency.
  - REST entry points defined with CrossOrigin, to grant access to market-web.
  - To be defined: Frequency of actualisation, visibility and security, resilience of service.
  - Possible extensions: History, Validation.
  - ...
- MARKET service
  - inventory for the current state of products in the market.
  - Request EUR value from the exchange service and apply it before storing in database.
  - REST entry points defined with CrossOrigin, to grant access to market-web.
  - to be defined: security, resilience, modifying a price of product and applied currency, update currency and cache.
  - ...
- MARKET web
  - user interface for handling products in the market.
  - to be defined: security, resilience, integrate WEB-UI in the module market with as example (com.github.eirslett::frontend-maven-plugin).

    - ...

Also, a point for reflection, the resilience of this environment, in case one of the components is not accessible. What is the impact and strategy to apply for the different scenarios.

## Workflow for a new product



For adding a new product to the market the following steps are executed:

- The user click on add new product.
- A popup window get visible. The form (Angular React-Form) keeps the button add disabled as long the form is not valid.
- The user submit the form.
- The MARKET-service receives the data and validate the DTO against the rules defined in the DTO (JSR-303).
- The MARKET-service request the EUR value for the given amount and currency from the exchange service. Requesting the calculation by Backend guarantee that the exchange rate could not be injected by the user. (A condition to not request values in EUR, could be used to reduce the number of requests). The calculation request is a SYNC operation, done with a Spring

Rest-Template in calling the REST-API of EXCHANGE-service.

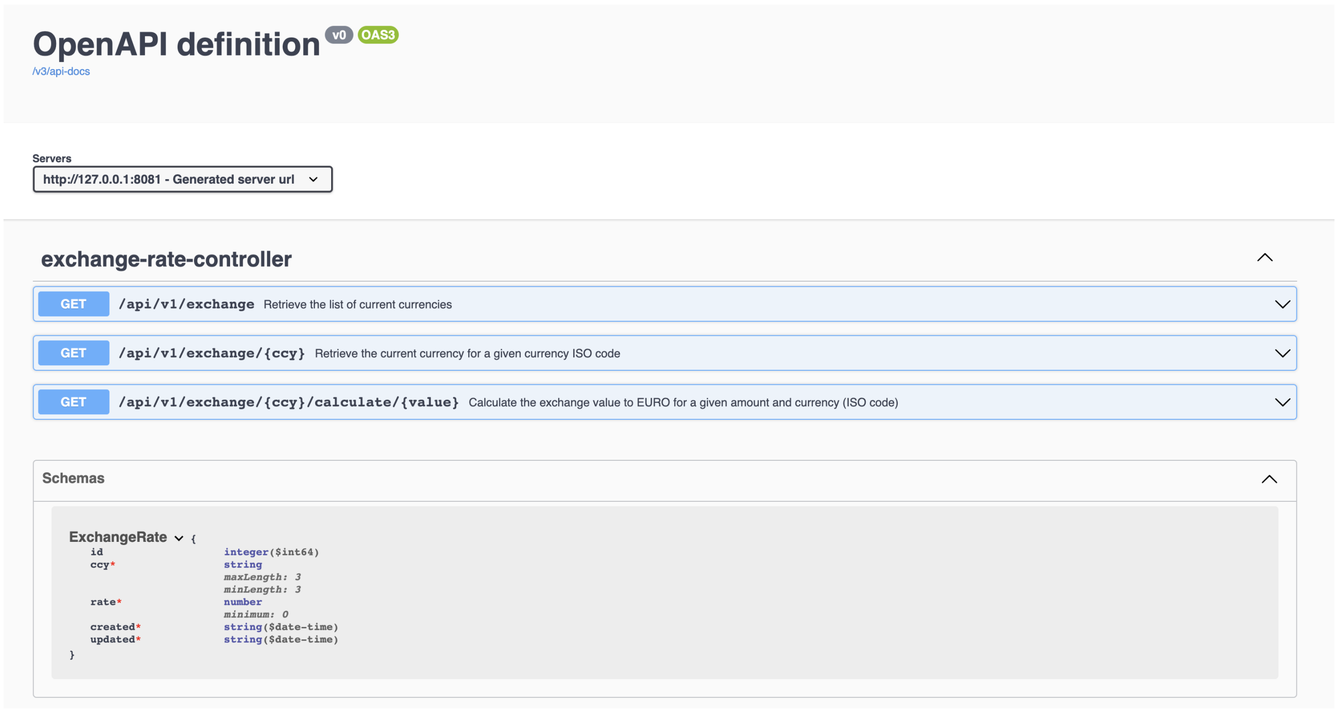
- The EXCHANGE-service calculate the exchange value, when the currency is known in his database. In case of a success, the result will be returned, in other case an error will be transmitted.
- In case of a success return, the information will be validated against the JSR-303 entity definition and stored in the database.
- In case of any error during the Backend-operation, the exception passed by a global @RestControllerAdvice and in case of a matching definition handled by it.

# REST Services

## Swagger Information

The java application (exchange & market) contains the [springdoc-openapi-ui] as dependency, for presenting the API REST documentation in Swagger format.

Exchange API View (http://127.0.0.1:8081/swagger-ui.html):


OpenAPI definition v0 OAS3
/v3/api-docs

Servers
http://127.0.0.1:8081 - Generated server url

exchange-rate-controller

GET /api/v1/exchange Retrieve the list of current currencies

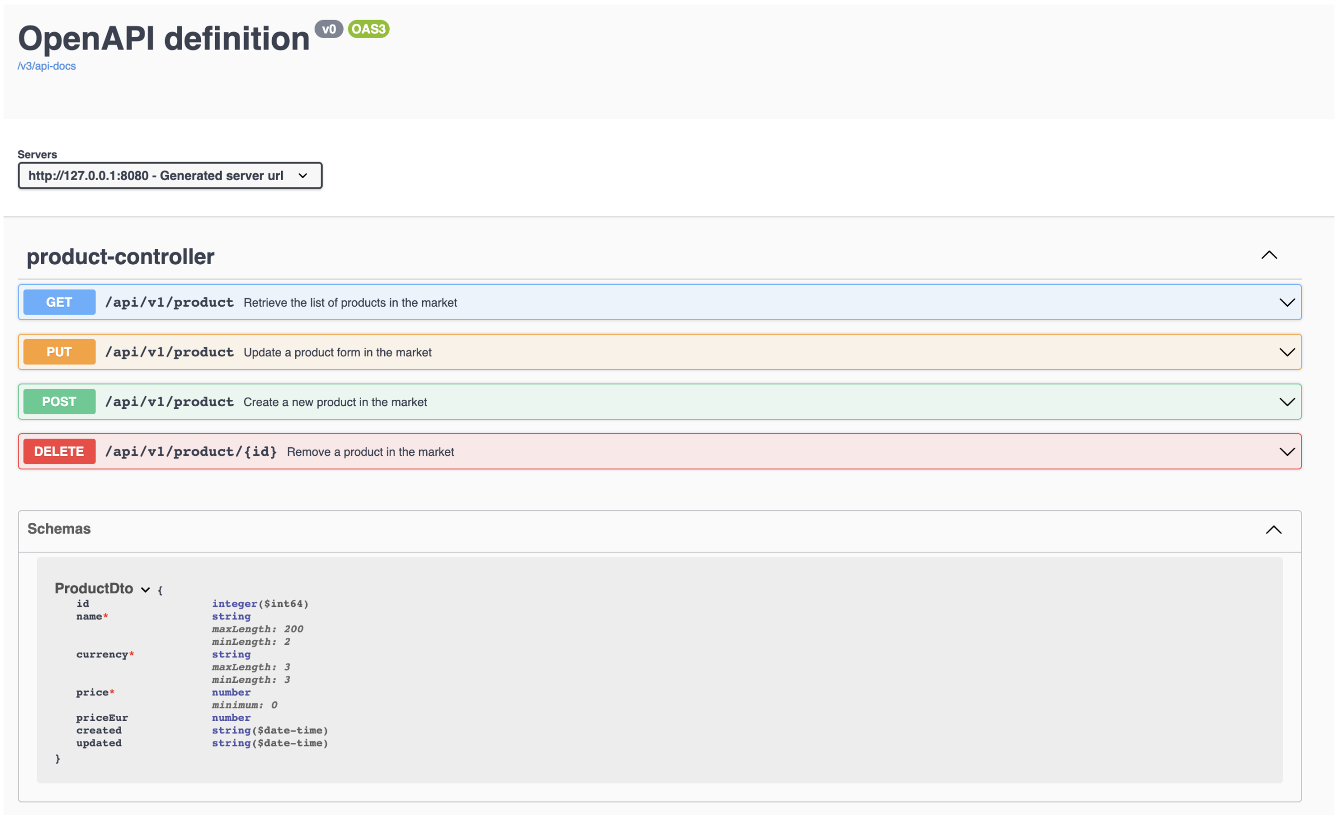
GET /api/v1/exchange/{ccy} Retrieve the current currency for a given currency ISO code

GET /api/v1/exchange/{ccy}/calculate/{value} Calculate the exchange value to EURO for a given amount and currency (ISO code)

Schemas

```
ExchangeRate {
    id              integer($int64)
    ccy*            string
                    maxLength: 3
                    minLength: 3
    rate*           number
                    minimum: 0
    created*        string($date-time)
    updated*        string($date-time)
}
```

Market API View (http://127.0.0.1:8080/swagger-ui.html):



OpenAPI definition v0 OAS3
/v3/api-docs

Servers
http://127.0.0.1:8080 - Generated server url

product-controller

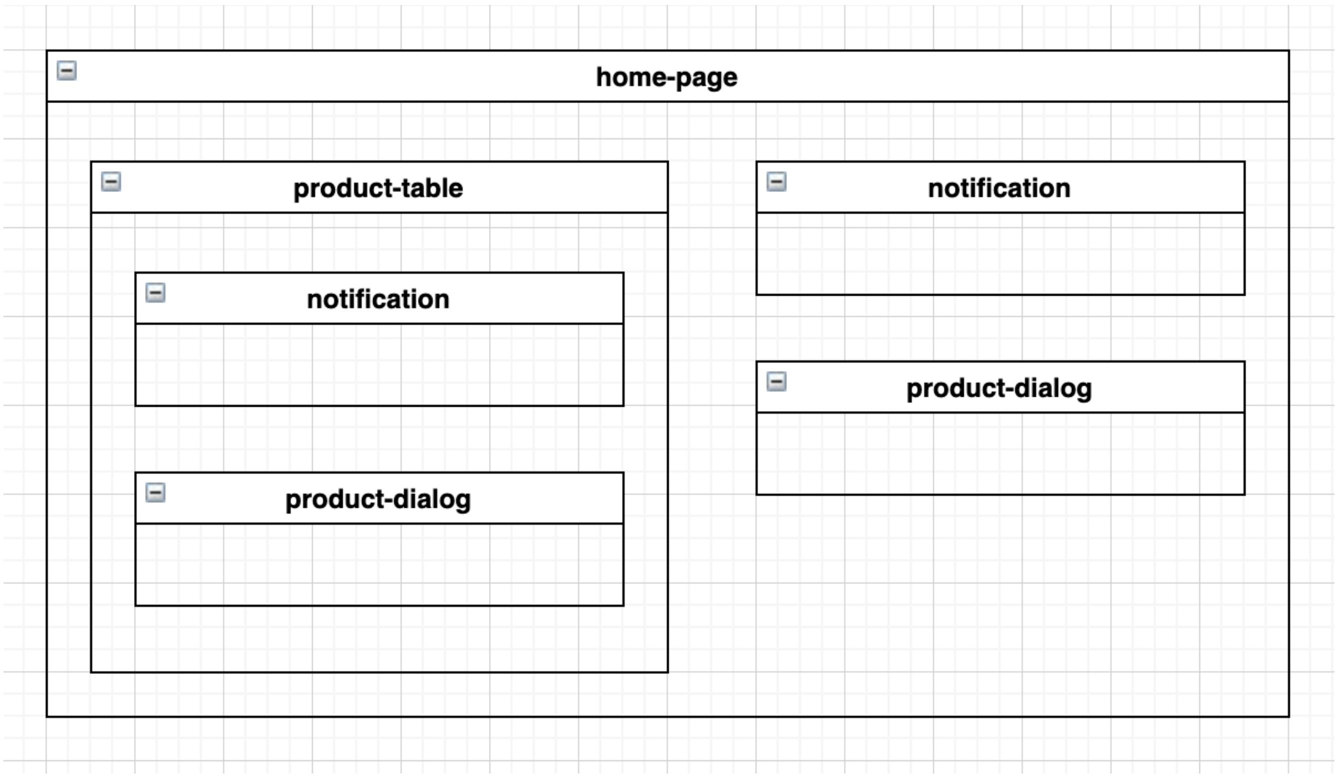
```
GET     /api/v1/product        Retrieve the list of products in the market
PUT     /api/v1/product        Update a product form in the market
POST    /api/v1/product        Create a new product in the market
DELETE  /api/v1/product/{id}   Remove a product in the market
```

Schemas

```
ProductDto {
    id              integer($int64)
    name*           string
                    maxLength: 200
                    minLength: 2
    currency*       string
                    maxLength: 3
                    minLength: 3
    price*          number
                    minimum: 0
    priceEur        number
    created         string($date-time)
    updated         string($date-time)
}
```

## Angular UI Components

The application is based on a simple component tree:



### home-page

The main component contain the components [product-table][product-dialog][notification]. The component displays the table and contains the action to add a new product by the product-dialog.

### product-table

The component contain the components [product-dialog][notification]. The component lists the products and contains the action to update a product with the product-dialog and delete a product.

### product-dialog

The component is based on a Angular React Form with validation rules and can be used to add or delete a product.

### notification

The component is a simple Wrapper for the Angular Material SnackBar to show messages (SUCCESS/ERROR) to the user.

## Stack Technology

### Java

- JDK 16
- Spring Boot 2.5.6
- Spring Boot JPA
- Spring Boot Web
- Spring Boot Validation
- Spring Boot Actuator
- Spring Boot Test
- Flyway
- H2 Database
- Mockito
- Springdoc Openapi

### Angular

- Angular 12
- Angular Material
- RxJs

# Annexe

## H2 Database + Flyway

The implementation for an H2 database in Spring has only a couple of configuration to do, as it is a POC we keep it light:

**Maven Dependencies (H2 + JPA)**

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

Configuration Spring Boot (database part in application.properties)

```
# Database (H2)
spring.datasource.url=jdbc:h2:file:./h2/market-dev-db
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
# http://localhost:8080/h2-console/
spring.h2.console.enabled=true
# JPA
spring.jpa.hibernate.ddl-auto=validate
spring.jpa.show-sql=true
```

**Maven Dependencies (Flyway)**

```
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-core</artifactId>
</dependency>
```

Configuration Spring Boot (Flyway part in application.properties)

```
# Flyway
spring.flyway.enabled=true
spring.flyway.user=sa
spring.flyway.password=password
spring.flyway.url=jdbc:h2:file:./h2/market-dev-db
```

**Database Schema**

The schema and database is stored under the project resource folder (resources/db/migration)

V1_0_0__database_schema.sql : containing the schema

V1_0_1__data.sql product-table: containing a set of example data

as the schema is created by flyway, in JPA we only validate the entity model against the database: (spring.jpa.hibernate.ddl-auto=validate).

As the application can be started with different Spring profiles, a configuration for the database or application port (server.port=8080) can be customized for purpose. An activation can be done in adding [-Dspring.profiles.active=dev], defining an environment variable [export spring_profiles_active=dev], or like in the POC, define the default profile in application.properties.

# Docker

The application can be run in Docker for this the Java and Angular modules are build by the multi stage scripts (Build Stage + Run Stage).

Run the Docker with: docker-compose up

Docker image: exchange

```
# BUILD STAGE
FROM maven:3.8.3-openjdk-16 AS maven

WORKDIR /usr/src/app
COPY . /usr/src/app
RUN mvn package

# RUN STAGE
FROM adoptopenjdk/openjdk16:alpine-jre

WORKDIR /opt/app
COPY --from=maven /usr/src/app/target/exchange-0.0.1-SNAPSHOT.jar /opt/app/
ENTRYPOINT ["java","-jar", "exchange-0.0.1-SNAPSHOT.jar"]
```

Docker image: market

```
# BUILD STAGE
FROM maven:3.8.3-openjdk-16 AS maven

WORKDIR /usr/src/app
COPY . /usr/src/app
RUN mvn package

# RUN STAGE
FROM adoptopenjdk/openjdk16:alpine-jre

WORKDIR /opt/app
COPY --from=maven /usr/src/app/target/market-0.0.1-SNAPSHOT.jar /opt/app/

ENV spring_profiles_active=docker

ENTRYPOINT ["java","-jar", "market-0.0.1-SNAPSHOT.jar"]
```

Docker image: market-web

```
# BUILD STAGE
FROM node:16.13.0-alpine as build
WORKDIR /usr/local/app
COPY ./ /usr/local/app/

RUN npm install
RUN npm run build

# RUN STAGE
FROM nginx:latest
COPY --from=build /usr/local/app/dist/market-web /usr/share/nginx/html
```

# Code Module Exchange

Entity (ExchangeRate.java)

```
/**
 * Entity for Database Table EXCHANGE_RATE
 */
@Entity
@Table(name = "EXCHANGE_RATE")
public class ExchangeRate {

    /**
     * Primary Key of Table
     */
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    /**
     * Currency represented in 3 chars
     */
    @Column(name = "ISO_CCY", unique = true)
    @Size(min = 3, max = 3, message = "Currency in exact 3 character length")
    @NotNull(message = "Currency is mandatory")
    private String ccy;

    /**
     * Exchange Rate handled as BigDecimal
     */
    @Column(name = "EXCHANGE_RATE")
    @Min(value = 0, message = "negative price not allowed")
    @NotNull(message = "Rate is mandatory")
    private BigDecimal rate;

    /**
     * Creation Date (Audit Value)
     */
    @Column(name = "CREATION_DATE")
    @NotNull(message = "Creation date is mandatory")
    private LocalDateTime created;

    /**
     * Modification Date (Audit Value)
     */
    @Column(name = "UPDATE_DATE")
    @NotNull(message = "Update date is mandatory")
    @Version
    private LocalDateTime updated;
```

Repository (ExchangeRateRepository.java)

```java
/**
 * Repository interface to Query the Exchange Rate from DB
 */
@Repository
public interface ExchangeRateRepository extends JpaRepository<ExchangeRate, Long> {

    /**
     * Retrieve an exchange rate from Database in case it exists, using Spring-JPA
naming convention for the method.
     * An other option @Query and writing the select statement in JPA Query language.
     * @param ccy currency for request
     * @return Optional exchange rate
     */
    Optional<ExchangeRate> findByCcy(String ccy);
}
```

Service (ExchangeRateServiceImpl.java)

```java
/**
 * Exchange Rate Service
 */
@Service
public class ExchangeRateServiceImpl implements ExchangeRateService {
    private final ExchangeRateRepository exchangeRateRepository;
    private final Logger logger = LoggerFactory.getLogger(ExchangeRateServiceImpl
.class);

    /**
     * Constructor with injected components
     * @param exchangeRateRepository Exchange Rate Repository
     */
    public ExchangeRateServiceImpl(ExchangeRateRepository exchangeRateRepository) {
        this.exchangeRateRepository = exchangeRateRepository;
    }

    /**
     * Get the list of exchange rates
     * @return List of exchange rates
     */
    @Override
    @Transactional(readOnly = true)
    public List<ExchangeRate> getCurrencies() {
        logger.info("get exchange list");
        return exchangeRateRepository.findAll();
    }

    /**
```

```java
     * Exchange a value in Euro
     * @param ccy base currency to exchange
     * @param value value to exchange
     * @return value exchanged in EUR
     */
    @Override
    @Transactional(readOnly = true)
    public BigDecimal calculateExchange(String ccy, BigDecimal value) {
        logger.info("calculate exchange [{}][{}]", ccy, value);
        var rate = getExchangeByCurrencyCode(ccy);
        return value.multiply(rate.getRate());
    }

    /**
     * Get an exchange rate for a requested currency
     * @param ccy currency in request
     * @return exchange rate for currency
     */
    @Override
    @Transactional(readOnly = true)
    public ExchangeRate getExchangeByCurrencyCode(String ccy) {
        logger.info("get exchange [{}]", ccy);
        return exchangeRateRepository.findByCcy(ccy)
                .orElseThrow(() -> new UnknownCurrencyException(ccy));
    }
}
```

Rest Controller (ExchangeRateController.java, RestExceptionHandler.java)

```java
/**
 * Rest Controller for Exchange Rate
 * - Entry Point: /api/v1/exchange
 * - CrossOrigin for UI to access to information
 */
@RestController
@CrossOrigin
@RequestMapping("/api/v1/exchange")
public class ExchangeRateController {

    private final ExchangeRateService exchangeRateService;

    public ExchangeRateController(ExchangeRateService exchangeRateService) {
        this.exchangeRateService = exchangeRateService;
    }

    /**
     * Get Exchange List
     * @return Exchange list (Name and Value)
     */
    @GetMapping
    @Operation(summary = "Retrieve the list of current currencies")
    public ResponseEntity<List<ExchangeRate>> getExchangeRateList() {
        return ResponseEntity.ok(exchangeRateService.getCurrencies());
    }

    /**
     * Get an EUR value for a given currency and value
     * @param ccy base currency
     * @param value base value
     * @return EUR value calculated
     */
    @GetMapping("/{ccy}/calculate/{value}")
    @Operation(summary = "Calculate the exchange value to EURO for a given amount and
currency (ISO code)")
    public ResponseEntity<BigDecimal> calculate(
            @Parameter(description = "Currency in ISO format")
            @PathVariable String ccy,
            @Parameter(description = "Value to change to EUR as Decimal")
            @PathVariable BigDecimal value) {
        var calculateExchange = exchangeRateService.calculateExchange(ccy, value);
        return ResponseEntity.ok(calculateExchange);
    }

    /**
     * Get an exchange value for given currency
     * @param ccy base currency
     * @return exchange value for the ccy
     */
    @GetMapping("/{ccy}")
    @Operation(summary = "Retrieve the current currency for a given currency ISO code
")
    public ResponseEntity<ExchangeRate> getExchangeRate(
            @Parameter(description = "Currency in ISO format")
            @PathVariable String ccy
    ) {
        var exchangeRate = exchangeRateService.getExchangeByCurrencyCode(ccy);
        return ResponseEntity.ok(exchangeRate);
    }
```

Rest Exception Handler (RestControllerAdvice)

```java
/**
 * Global Exception Handler for as gate for the Exception and message transmitted by
the Rest Request
 */
@RestControllerAdvice
public class RestExceptionHandler extends ResponseEntityExceptionHandler {

    /**
     * Handler for an unknown currency
     * @param ex UnknownCurrencyException handled
     * @param request HTTP request
     * @return HTTP Response (400 - Bad request)
     */
    @ExceptionHandler(value = {UnknownCurrencyException.class})
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public ResponseEntity<Object> handleUnknownCurrencyException(RuntimeException ex,
WebRequest request) {
        return handleExceptionInternal(ex, ex.getMessage(),
                httpHeadersSupplier.get(), HttpStatus.BAD_REQUEST, request);
    }

    /**
     * Creating a Header with CrossOrigin
     */
    private final Supplier<HttpHeaders> httpHeadersSupplier = () -> {
        var header = new HttpHeaders();
        header.setAccessControlAllowMethods(List.of(HttpMethod.POST));
        header.setAccessControlAllowHeaders(List.of(HttpHeaders.CONTENT_TYPE));
        header.setAllow(Set.of(HttpMethod.GET, HttpMethod.HEAD, HttpMethod.POST,
HttpMethod.PUT, HttpMethod.DELETE, HttpMethod.OPTIONS, HttpMethod.PATCH));
        return header;
    };
}
```

Configuration (application.properties)

```
# Server
server.port=8081
# Database (H2)
spring.datasource.url=jdbc:h2:file:./h2/exchange-dev-db
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
# http://localhost:8081/h2-console/
spring.h2.console.enabled=true
# JPA
spring.jpa.hibernate.ddl-auto=validate
spring.jpa.show-sql=true
# Flyway
spring.flyway.enabled=true
spring.flyway.user=sa
spring.flyway.password=password
spring.flyway.url=jdbc:h2:file:./h2/exchange-dev-db
```

Flyway DB Scripts

```
CREATE TABLE IF NOT EXISTS EXCHANGE_RATE(
    ID            BIGINT AUTO_INCREMENT,
    ISO_CCY       VARCHAR(3) UNIQUE NOT NULL,
    EXCHANGE_RATE DECIMAL(19, 6) NOT NULL,
    CREATION_DATE TIMESTAMP NOT NULL,
    UPDATE_DATE TIMESTAMP NOT NULL
);
---
```

Unit Test

```
/**
 * Echange Rate Test Suite
 */
@SpringBootTest
class ExchangeRangeServiceTest {

    @Autowired
    private ExchangeRateService exchangeRateService;

    /**
     * Retrieve a list of current exchange rate (SUCCESS)
     */
    @Test
    public void getCurrenciesTest() {
        List<ExchangeRate> currencies = exchangeRateService.getCurrencies();
        assertFalse(currencies.isEmpty(), "No currencies in the list (Empty List)");
    }

    /**
     * Retrieve a currency for an existing CCY Code (SUCCESS)
     */
    @Test
    public void getByExchangeCodeSuccessTest() {
        var exchange = exchangeRateService.getExchangeByCurrencyCode("USD");
        assertEquals(new BigDecimal(1.3, MathContext.DECIMAL32), exchange.getRate());
    }

    /**
     * Retrieve a currency for an non-existing CCY Code (ERROR)
     */
    @Test
    public void getByExchangeCodeErrorTest() {
        Exception exception = assertThrows(UnknownCurrencyException.class, () ->
exchangeRateService.getExchangeByCurrencyCode("FYI"));
        assertNotNull(exception, "No exception thrown");
        assertNotNull(exception.getMessage(), "No message in the exception");
        assertTrue(exception.getMessage().contains("FYI"), "Currency not included in
message");
    }

    /**
     * calculating an exchange for a given currency and value (SUCCESS)
     */
    @Test
    public void calculateExchangeTest() {
        var result = exchangeRateService.calculateExchange("USD", new BigDecimal(2));
        assertEquals(new BigDecimal(2.6, MathContext.DECIMAL32), result, "value not
correct exchanged");
    }

    /**
     * calculating an exchange for a given currency and value (ERROR)
     */
    @Test
    public void calculateExchangeUnknownCurrencyTest() {
        Exception exception = assertThrows(UnknownCurrencyException.class, () ->
exchangeRateService.calculateExchange("FYI", new BigDecimal(2)));
        assertNotNull(exception, "No exception thrown");
        assertNotNull(exception.getMessage(), "No message in the exception");
        assertTrue(exception.getMessage().contains("FYI"), "Currency not included in
message");
    }
}
```

# Code Module market

Entity (ExchangeRate.java)

```
/**
 * Entity for Database Table PRODUCT
 */
@Entity
public class Product {

    /**
     * Primary Key of Table
     */
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    /**
     * Product name
     */
    @Column(name = "PRODUCT_NAME")
    @NotBlank(message = "Name is mandatory")
    @Size(min = 2, max = 200, message = "Product name must be at least 2 and maximal
200 character length")
    private String name;

    /**
     * Currency of product represented in 3 chars
     */
    @Column(name = "CURRENCY")
    @NotBlank(message = "Currency is mandatory")
    @Size(min = 3, max = 3, message = "Currency in exact 3 character length")
    private String currency;

    /**
     * Price in Product currency
     */
    @Column(name = "PRICE")
    @NotNull(message = "Price is mandatory")
    @Min(value = 0, message = "negative price not allowed")
    private BigDecimal price;

    /**
     * Price in EUR currency
     */
    @Column(name = "PRICE_EUR")
    @NotNull(message = "Price is mandatory")
    private BigDecimal priceEur;

    /**
```

```
     * Creation Date (Audit Value)
     */
    @Column(name = "CREATION_DATE")
    @NotNull(message = "Creation date is mandatory")
    private LocalDateTime created;

    /**
     * Modification Date (Audit Value)
     */
    @Column(name = "UPDATE_DATE")
    @NotNull(message = "Update date is mandatory")
    @Version
    private LocalDateTime updated;
```

Repository (ProductRepository.java)

```
/**
 * Repository interface to Query the Product from DB
 */
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {
}
```

Service(CurrencyRestServiceImpl.java, ProductServiceImpl.java)

```
@Service
public class CurrencyRestServiceImpl implements com.bindstone.acm.market.service
.CurrencyRestService {
    private final RestTemplate restTemplate;
    private final Logger logger = LoggerFactory.getLogger(CurrencyRestServiceImpl
.class);
    @Value("${acm.exchange.server}")
    private String currencyServer;

    /**
     * Currency Rest Service
     * The implementation use a Spring Rest Template to retrieve the Data from the
exchange Service.
     * An alternative to the Rest Template could be Retrofit.
     */
    public CurrencyRestServiceImpl() {
        this.restTemplate = new RestTemplate();
    }

    /**
     * Request calculation of exchange by external service
     * @param ccy product currency
     * @param value product value
     * @return EUR value calculated
     */
    @Override
    public BigDecimal calculateExchange(String ccy, BigDecimal value) {

        var url = String.format("%s/api/v1/exchange/%s/calculate/%s", currencyServer,
ccy, value);
        logger.info("Request: {}", url);
        ResponseEntity<BigDecimal> response = null;
        try {
            response = restTemplate.getForEntity(url, BigDecimal.class);
        } catch (Exception e) {
            logger.error(String.format("Server error for request: %s", url), e);
            throw new ExchangeRateServiceConnectionException();
        }
        if (response.getStatusCode().is2xxSuccessful()) {
            return response.getBody();
        } else {
            logger.error(String.format("Error in calculating exchange for [%s][%f]",
ccy, value));
            throw new ExchangeRateRequestException(ccy, value);
        }
    }
}
```

```
/**
```

```
 * Product Service
 */
@Service
public class ProductServiceImpl implements ProductService {
    private final CurrencyRestService currencyRestService;
    private final ProductRepository productRepository;
    private final ProductMapper productMapper;
    final Logger logger = LoggerFactory.getLogger(ProductServiceImpl.class);

    /**
     * Constructor injecting the dependencies and create the Product Mapper for the
DTO
     * @param currencyRestService Currency Rest Service to communicate with external
service
     * @param productRepository Product Repository to query the Database
     */
    public ProductServiceImpl(CurrencyRestService currencyRestService,
ProductRepository productRepository) {
        this.currencyRestService = currencyRestService;
        this.productRepository = productRepository;
        this.productMapper = new ProductMapper();
    }

    /**
     * Get the Product list from database
     * @return List of products
     */
    @Override
    @Transactional(readOnly = true)
    public List<Product> getProductList() {
        logger.info("Get all products");
        return productRepository.findAll();
    }

    /**
     * Get the Product list from database wrapped with DTO transformation
     * @return List of products (DTO)
     */
    @Override
    @Transactional(readOnly = true)
    public List<ProductDto> getProductDtoList() {
        return getProductList().stream().map(x -> productMapper.toDto(x)).collect
(Collectors.toList());
    }

    /**
     * Create a new Product, calculate the exchange value, and store in Database
     * @param product new Product
     * @return created Product (include ID and timestamps)
     */
    @Override
    @Transactional
    public Product create(Product product) {
        logger.info("Create new product");
        var euro = currencyRestService.calculateExchange(product.getCurrency(),
product.getPrice());
        product.setCreated(LocalDateTime.now());
        product.setPriceEur(euro);
        return productRepository.save(product);
    }

    /**
     * Create a new Product, calculate the exchange value, and store in Database,
wrapped with DTO transformation
     * @param product new Product as DTO
     * @return created Product (include ID and timestamps) (DTO)
     */
    @Override
    @Transactional
    public ProductDto create(ProductDto product) {
        var jpa = productMapper.fromDto(product);
        return productMapper.toDto(create(jpa));
    }

    /**
     * Update Product, calculate the exchange value in case the currency or value
changed,
     * and store in Database.
     * @param product Product
     * @return Product (include ID and timestamps)
     */
    @Override
    @Transactional
    public Product update(Product product) {
        logger.info("Update product");
        // Business Rules validation
        // - Example, past, current exchange rate...
        var oldProduct = productRepository.findById(product.getId()).orElseThrow(() ->
new UnknownProductException(product.getId()));
        if (!oldProduct.getPrice().equals(product.getPrice()) || !oldProduct
.getCurrency().equals(product.getCurrency())) {
            var euro = currencyRestService.calculateExchange(product.getCurrency(),
product.getPrice());
            product.setPriceEur(euro);
        }
        return productRepository.save(product);
    }

    /**
     * Update Product, calculate the exchange value in case the currency or value
changed,
     * and store in Database, wrapped with DTO transformation.
     * @param product Product as DTO
     * @return Product (include ID and timestamps) (DTO)
     */
    @Override
    @Transactional
    public ProductDto update(ProductDto product) {
        var jpa = productMapper.fromDto(product);
        return productMapper.toDto(update(jpa));
    }

    /**
     * Delete (Remove) a product from the market
     * @param id Product id
     */
    @Override
    @Transactional
    public void delete(Long id) {
        logger.info("Delete product");
        var product = productRepository.findById(id).orElseThrow(() -> new
UnknownProductException(id));
        productRepository.delete(product);
    }
}
```

DTO Mapper

```
public class ProductDto {

    private Long id;

    @NotBlank(message = "Name is mandatory")
    @Size(min = 2, max = 200, message = "Product name must be at least 2 and maximal
200 character length")
    private String name;

    @NotBlank(message = "Currency is mandatory")
    @Size(min = 3, max = 3, message = "Currency in exact 3 character length")
    private String currency;

    @NotNull(message = "Price is mandatory")
    @Min(value = 0, message = "negative price not allowed")
    private BigDecimal price;

    private BigDecimal priceEur;

    private LocalDateTime created;

    private LocalDateTime updated;
```

```java
/**
 * Transformation of the Entity (JPA) Product object to a external REST Object.
 * This could be replaced by an implementation of as example  the framework MapStruct.
 */
public class ProductMapper implements DtoMapper<Product, ProductDto>{
    /**
     * Transform JPA -> DTO (Product)
     * @param obj JPA Product object
     * @return DTO Product object
     */
    @Override
    public ProductDto toDto(Product obj) {
        if(Objects.isNull(obj)) {
            return null;
        }
        ProductDto rtn = new ProductDto();
        rtn.setId(obj.getId());
        rtn.setName(obj.getName());
        rtn.setCurrency(obj.getCurrency());
        rtn.setPrice(obj.getPrice());
        rtn.setPriceEur(obj.getPriceEur());
        rtn.setCreated(obj.getCreated());
        rtn.setUpdated(obj.getUpdated());
        return rtn;
    }

    /**
     * Transform DTO -> JPA (Product)
     * @param obj DTO Product object
     * @return JPA Product object
     */
    @Override
    public Product fromDto(ProductDto obj) {
        if(Objects.isNull(obj)) {
            return null;
        }
        Product rtn = new Product();
        rtn.setId(obj.getId());
        rtn.setName(obj.getName());
        rtn.setCurrency(obj.getCurrency());
        rtn.setPrice(obj.getPrice());
        rtn.setPriceEur(obj.getPriceEur());
        rtn.setCreated(obj.getCreated());
        rtn.setUpdated(obj.getUpdated());
        return rtn;
    }
}
```

Rest Controller (ProductController.java, RestExceptionHandler.java)

```java
/**
 * Rest Controller for the products in market
 * - Entry Point: /api/v1/product
 * - CrossOrigin for UI to access to information
 */
@RestController
@CrossOrigin
@RequestMapping("/api/v1/product")
public class ProductController {

    private final ProductService productService;

    public ProductController(ProductService productService) {
        this.productService = productService;
    }

    /**
     *
     * @return
     */
    @GetMapping
    @Operation(summary = "Retrieve the list of products in the market")
    public ResponseEntity<List<ProductDto>> getProductDtoList() {
        return ResponseEntity.ok(productService.getProductDtoList());
    }

    /**
     *
     * @param product
     * @return
     */
    @PutMapping
    @Operation(summary = "Update a product form in the market")
    public ResponseEntity<ProductDto> updateProduct(@Valid @RequestBody ProductDto
product) {
        return ResponseEntity.ok(productService.update(product));
    }

    /**
     *
     * @param product
     * @return
     */
    @PostMapping
    @Operation(summary = "Create a new product in the market")
    public ResponseEntity<ProductDto> createProduct(@Valid @RequestBody ProductDto
product) {
        return ResponseEntity.ok(productService.create(product));
    }

    /**
```

```java
     *
     * @param id
     * @return
     */
    @DeleteMapping("/{id}")
    @Operation(summary = "Remove a product in the market")
    public ResponseEntity<Void> deleteProduct(
            @Parameter(description = "Product ID")
            @PathVariable Long id
    ) {
        productService.delete(id);
        return ResponseEntity.ok().build();
    }
}
```

```java
/**
 * Global Exception Handler for as gate for the Exception and message transmitted by
the Rest Request
 */
@RestControllerAdvice
public class RestExceptionHandler extends ResponseEntityExceptionHandler {

    /**
     * Handler for an error response from the exchange rate service
     * @param ex ExchangeRateRequestException
     * @param request HTTP request
     * @return HTTP Response (400 - Bad request)
     */
    @ExceptionHandler(value = {ExchangeRateRequestException.class})
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public ResponseEntity<Object> handleExchangeRateRequestException(RuntimeException
ex, WebRequest request) {
        return handleExceptionInternal(ex, ex.getMessage(),
                httpHeadersSupplier.get(), HttpStatus.BAD_REQUEST, request);
    }

    /**
     * Handler for the communication errors to the exchange rate service
     * @param ex ExchangeRateServiceConnectionException
     * @param request HTTP request
     * @return HTTP Response (503 - Service Unavailable)
     */
    @ExceptionHandler(value = {ExchangeRateServiceConnectionException.class})
    @ResponseStatus(HttpStatus.SERVICE_UNAVAILABLE)
    public ResponseEntity<Object> handleExchangeRateServiceConnectionException
(RuntimeException ex, WebRequest request) {
        return handleExceptionInternal(ex, ex.getMessage(),
                httpHeadersSupplier.get(), HttpStatus.SERVICE_UNAVAILABLE, request);
    }

    /**
     * Creating a Header with CrossOrigin
     */
    private final Supplier<HttpHeaders> httpHeadersSupplier = () -> {
        var header = new HttpHeaders();
        header.setAccessControlAllowMethods(List.of(HttpMethod.POST));
        header.setAccessControlAllowHeaders(List.of(HttpHeaders.CONTENT_TYPE));
        header.setAllow(Set.of(HttpMethod.GET, HttpMethod.HEAD, HttpMethod.POST,
HttpMethod.PUT, HttpMethod.DELETE, HttpMethod.OPTIONS, HttpMethod.PATCH));
        return header;
    };

}
```

Configuration (application.properties)

```
# Server
server.port=8080
# Database (H2)
spring.datasource.url=jdbc:h2:file:./h2/market-dev-db
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
# http://localhost:8080/h2-console/
spring.h2.console.enabled=true
# JPA
spring.jpa.hibernate.ddl-auto=validate
spring.jpa.show-sql=true
# Flyway
spring.flyway.enabled=true
spring.flyway.user=sa
spring.flyway.password=password
spring.flyway.url=jdbc:h2:file:./h2/market-dev-db
# application
acm.exchange.server=http://127.0.0.1:8081
```

Flyway DB Scripts

```sql
CREATE TABLE IF NOT EXISTS PRODUCT (
   ID BIGINT AUTO_INCREMENT,
   PRODUCT_NAME VARCHAR(200) NOT NULL,
   CURRENCY VARCHAR(3) NOT NULL,
   PRICE DECIMAL(19,6) NOT NULL,
   PRICE_EUR DECIMAL(19,6)  NOT NULL,
   CREATION_DATE TIMESTAMP NOT NULL,
   UPDATE_DATE TIMESTAMP NOT NULL
);
---
```

Unit Test

```java
/**
 * Product Test Suite, including a Mockup for connection with the external service
EXCHANGE-SERVICE
 */
@SpringBootTest
class ProductServiceTest {

    @Autowired
    ProductRepository productRepository;

    CurrencyRestService currencyRestService;
    ProductService productService;

    /**
     * INSERT PRODUCT // SUCCESS
     */
    @Test
    public void getCurrenciesSuccessTest() {
        currencyRestService = Mockito.mock(CurrencyRestService.class);
        productService = new ProductServiceImpl(currencyRestService,
productRepository);

        Mockito.when(currencyRestService.calculateExchange("USD", BigDecimal.valueOf(
1))).thenReturn(BigDecimal.valueOf(3.5));

        var product = new Product();
        product.setName("Test Product");
        product.setPrice(BigDecimal.valueOf(1));
        product.setCurrency("USD");
        var calculate = productService.create(product);
        assertEquals(BigDecimal.valueOf(3.5), product.getPriceEur());
    }

    /**
     * Connection ERROR to external service
     */
    @Test
    public void getCurrenciesConnectionErrorTest() {
        currencyRestService = Mockito.mock(CurrencyRestService.class);
        productService = new ProductServiceImpl(currencyRestService,
productRepository);

        Mockito.when(currencyRestService.calculateExchange("USD", BigDecimal.valueOf(
1))).thenThrow(new ExchangeRateServiceConnectionException());

        var product = new Product();
        product.setName("Test Product");
        product.setPrice(BigDecimal.valueOf(1));
        product.setCurrency("USD");

        Exception exception = assertThrows(ExchangeRateServiceConnectionException
.class, () -> {
            var calculate = productService.create(product);
        });
    }

    /**
     * INVALID Data transmitted to external service
     */
    @Test
    public void getCurrenciesDataErrorTest() {
        currencyRestService = Mockito.mock(CurrencyRestService.class);
        productService = new ProductServiceImpl(currencyRestService,
productRepository);

        Mockito.when(currencyRestService.calculateExchange("FYI", BigDecimal.valueOf(
1))).thenThrow(new ExchangeRateRequestException("FYI", BigDecimal.valueOf(1)));

        var product = new Product();
        product.setName("Test Product");
        product.setPrice(BigDecimal.valueOf(1));
        product.setCurrency("FYI");

        Exception exception = assertThrows(ExchangeRateRequestException.class, () -> {
            var calculate = productService.create(product);
        });
    }

    /**
     * Invalid DATA to save to Database (name to short)
     */
    @Test
    public void getCurrenciesConstraintErrorTest() {
        currencyRestService = Mockito.mock(CurrencyRestService.class);
        productService = new ProductServiceImpl(currencyRestService,
productRepository);

        Mockito.when(currencyRestService.calculateExchange("USD", BigDecimal.valueOf(
1))).thenReturn(BigDecimal.valueOf(3.5));

        var product = new Product();
        product.setName("T");
        product.setPrice(BigDecimal.valueOf(1));
        product.setCurrency("USD");

        Exception exception = assertThrows(ConstraintViolationException.class, () -> {
            var calculate = productService.create(product);
        });

        assertTrue(exception.getMessage().contains("Product name must be at least 2
and maximal 200 character"), "Invalid constraint message");
    }
}
```

## Code Module market-web

app.module.ts

```
import {NgModule} from '@angular/core';
import {BrowserModule} from '@angular/platform-browser';
import {AppComponent} from './app.component';
import {HomePageComponent} from './component/home-page/home-page.component';
import {ProductTableComponent} from './component/product-table/product-
table.component';
import {ProductDialogComponent} from './component/product-dialog/product-
dialog.component';
import {BrowserAnimationsModule} from '@angular/platform-browser/animations';
import {MatToolbarModule} from "@angular/material/toolbar";
import {MatButtonModule} from "@angular/material/button";
import {MatCardModule} from "@angular/material/card";
import {MatIconModule} from "@angular/material/icon";
import {MatTableModule} from "@angular/material/table";
import {MatDialogModule} from "@angular/material/dialog";
import {MatFormFieldModule} from "@angular/material/form-field";
import {ReactiveFormsModule} from "@angular/forms";
import {MatInputModule} from "@angular/material/input";
import {MatSelectModule} from "@angular/material/select";
import {HttpClientModule} from "@angular/common/http";
import {NotificationComponent} from './component/notification/notification.component';
import {MatSnackBarModule} from "@angular/material/snack-bar";

@NgModule({
  declarations: [
    AppComponent,
    HomePageComponent,
    ProductTableComponent,
    ProductDialogComponent,
    NotificationComponent,
  ],
  imports: [
    BrowserModule,
    BrowserAnimationsModule,
    HttpClientModule,
    MatToolbarModule,
    MatButtonModule,
    MatCardModule,
    MatIconModule,
    MatTableModule,
    MatDialogModule,
    MatFormFieldModule,
    MatSnackBarModule,
    ReactiveFormsModule,
    MatInputModule,
    MatSelectModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {
}
```

Service (currency.service.ts, product-service.ts)

```
@Injectable({
  providedIn: 'root'
})
export class CurrencyService {
  baseUrl = "http://127.0.0.1:8081/api/v1/exchange/";

  constructor(private http: HttpClient) {
  }

  public getCurrencyList(): Observable<any> {
    return this.http.get(this.baseUrl)
      .pipe(
        map(res => (res as Array<any>)
          .map(exchange => exchange.ccy
          )));
  }
}
```

```
@Injectable({
  providedIn: 'root'
})
export class ProductService {

  baseUrl = "http://127.0.0.1:8080/api/v1/product/"

  constructor(private http: HttpClient) {
  }

  public getProductList(): Observable<any> {
    return this.http.get(this.baseUrl)
      .pipe(
        map(res => (res as Array<Product>)
        )
      );
  }

  public deleteProduct(id: number): Observable<any> {
    return this.http.delete(`${this.baseUrl}${id}`);
  }

  addProduct(product: Product) {
    return this.http.post(this.baseUrl, product);
  }

  updateProduct(product: Product) {
    return this.http.put(this.baseUrl, product);
  }
}
```

Component (home-page)

```
export class HomePageComponent implements OnInit {

  @ViewChild(ProductTableComponent) productTable!: ProductTableComponent;

  constructor(private productService: ProductService, private notification:
NotificationComponent, public productDialog: MatDialog) {
  }

  public addProduct() {
    this.productDialog.open(ProductDialogComponent, {
      width: '600px',
      height: '400px'
    }).afterClosed().subscribe(product => {
      if (product != null && product != "") {
        this.productService.addProduct(product)
          .subscribe(result => {
              this.notification.ok(`Product ${product.name} added to market`);
              this.productTable.refresh();
            },
            error => {
              if (error && error.error) {
                this.notification.error(error.error);
              } else {
                this.notification.error("Error adding Product, please try again");
              }
            })
      }
    });
  }

  ngOnInit(): void {
  }
}
```

```
<mat-toolbar>
  <span>ACM Market</span>
</mat-toolbar>

<div class="product-container">
  <div>
    <div class="product-title">
      <h1 class="title">Current Product List</h1>
      <button (click)="addProduct()" class="add-button" color="primary" mat-stroked-
button>
        Add new product
        <mat-icon class="add-button-icon">add_circle</mat-icon>
      </button>
    </div>
    <app-product-table></app-product-table>
  </div>
</div>
```

Component (product-dialog)

```
export class ProductDialogComponent implements OnInit {
  public productForm = new FormGroup({
    //name: new FormControl(null, [Validators.required, Validators.minLength(2),
Validators.maxLength(200)]),
    name: new FormControl(null, [Validators.required, Validators.maxLength(200)]),
    currency: new FormControl(null, [Validators.required]),
    price: new FormControl(null, [Validators.required, Validators.min(0.01),
Validators.max(5000)])
  })

  currencies: String[];

  constructor(public dialogRef: MatDialogRef<ProductDialogComponent>,
              private currencyService: CurrencyService,
              @Inject(MAT_DIALOG_DATA) public data: Product) {
    this.currencies = ["EUR"];
  }

  public submit(form: any) {
    let product = (form.value as Product);
    if (this.data) {
      product.id = this.data.id;
      product.created = this.data.created;
      product.updated = this.data.updated;
    }
    this.dialogRef.close(product);

  }

  ngOnInit(): void {
    if (this.data != null) {
      this.productForm.setValue({
        name: this.data.name,
        currency: this.data.currency,
        price: this.data.price
      })
    }
    this.currencyService.getCurrencyList().subscribe(result => this.currencies =
result.sort());
  }
}
```

```
<h2 mat-dialog-title>{{data ? 'Update product' : 'Add new product'}}</h2>
<form (ngSubmit)="submit(productForm)" [formGroup]="productForm">
  <mat-dialog-content>
    <div class="row">
      <mat-form-field>
        <mat-label>Product name</mat-label>
        <input formControlName="name" matInput required type="text"/>
        <mat-error *ngIf="(productForm.get('name')?.errors?.required)">Field is
required</mat-error>
        <mat-error *ngIf="(productForm.get('name')?.errors?.minlength)">Please at
least 2 chars</mat-error>
        <mat-error *ngIf="(productForm.get('name')?.errors?.maxlength)">Please not
more than 200 chars</mat-error>
      </mat-form-field>
    </div>
    <div class="row">
      <mat-form-field>
        <mat-label>Price</mat-label>
        <input formControlName="price" matInput required step="0.01" type="number"/>
        <mat-error *ngIf="(productForm.get('price')?.errors?.required)">Field is
required</mat-error>
        <mat-error *ngIf="(productForm.get('price')?.errors?.min)">Price must be
greater than 0</mat-error>
        <mat-error *ngIf="(productForm.get('price')?.errors?.max)">Price must be less
than 5000</mat-error>
      </mat-form-field>
    </div>
    <div class="row">
      <mat-form-field>
        <mat-label>Currency (Price)</mat-label>
        <mat-select formControlName="currency" required>
          <mat-option *ngFor="let ccy of currencies" [value]="ccy">
            {{ccy}}
          </mat-option>
        </mat-select>
        <mat-error *ngIf="(productForm.get('currency')?.errors?.required)">Field is
required</mat-error>
      </mat-form-field>
    </div>
  </mat-dialog-content>
  <mat-dialog-actions align="end">
    <button mat-button mat-dialog-close>Cancel</button>
    <button [disabled]="!productForm.valid" color="primary" mat-button
            type="submit">{{data ? 'Update' : 'Add'}}</button>
  </mat-dialog-actions>
</form>
```

Component (product-table)

```
export class ProductTableComponent implements OnInit, OnDestroy {
  displayedColumns: string[] = ['name', 'price', 'currency', 'priceEur', 'action'];
  dataSource: MatTableDataSource<Product>;
  private productSubscription: Subscription | undefined;
  private productObservable: Observable<Product[]> | undefined;
  private products: Product[];

  constructor(private productService: ProductService, private notification:
NotificationComponent, public productDialog: MatDialog) {
    this.products = [];
    this.dataSource = new MatTableDataSource<Product>(this.products);
  }

  ngOnInit(): void {
    this.refresh();
  }

  public refresh() {
    this.productObservable = this.productService.getProductList();
    this.productSubscription = this.productObservable.subscribe(value => {
      this.products = value;
      this.dataSource = new MatTableDataSource<Product>(this.products);
    })
  }

  delete(product: Product) {
    this.productService.deleteProduct(product.id).subscribe(x => this.refresh());
  }

  edit(product: Product) {
    this.productDialog.open(ProductDialogComponent, {
      width: '600px',
      height: '400px',
      data: product
    }).afterClosed().subscribe(product => {
      if (product != null && product != "") {
        this.productService.updateProduct(product)
          .subscribe(result => this.refresh(),
            error => {
              if (error && error.error) {
                this.notification.error(error.error);
              } else {
                this.notification.error("Error adding Product, please try again");
              }
            });
      }
    });
  }

  ngOnDestroy(): void {
    if (this.productSubscription) {
      this.productSubscription.unsubscribe();
    }
  }
}
```

```
<div class="product-list-div">
  <mat-card>
    <mat-card-content>
      <table [dataSource]="dataSource" class="mat-elevation-z8" mat-table>

        <ng-container matColumnDef="name">
          <th *matHeaderCellDef mat-header-cell>Name</th>
          <td *matCellDef="let product" mat-cell> {{product.name}} </td>
        </ng-container>

        <ng-container matColumnDef="currency">
          <th *matHeaderCellDef mat-header-cell>Currency</th>
          <td *matCellDef="let product" mat-cell> {{product.currency}} </td>
        </ng-container>

        <ng-container matColumnDef="price">
          <th *matHeaderCellDef mat-header-cell>Price</th>
          <td *matCellDef="let product" mat-cell> {{product.price}} </td>
        </ng-container>

        <ng-container matColumnDef="priceEur">
          <th *matHeaderCellDef mat-header-cell>Price (EUR)</th>
          <td *matCellDef="let product" mat-cell> {{product.priceEur}} </td>
        </ng-container>

        <ng-container matColumnDef="action">
          <th *matHeaderCellDef mat-header-cell></th>
          <td *matCellDef="let product" mat-cell>
            <button (click)="edit(product)" class="icon-button" color="primary" mat-
mini-fab>
              <mat-icon>create</mat-icon>
            </button>
            <button (click)="delete(product)" class="icon-button" color="warn" mat-
mini-fab>
              <mat-icon>delete</mat-icon>
            </button>
          </td>
        </ng-container>

        <tr *matHeaderRowDef="displayedColumns" mat-header-row></tr>
        <tr *matRowDef="let row; columns: displayedColumns;" mat-row></tr>

      </table>
    </mat-card-content>
  </mat-card>
</div>
```

Component (notification)

```
export class NotificationComponent implements OnInit {

  constructor(private snackBar: MatSnackBar) {
  }

  ngOnInit(): void {
  }

  public error(message: string) {
    this.snackBar.open(message, undefined, {
      horizontalPosition: 'center',
      verticalPosition: 'top',
      panelClass: 'snackbar-error',
      duration: 5000
    })
  }

  public ok(message: string) {
    this.snackBar.open(message, undefined, {
      horizontalPosition: 'center',
      verticalPosition: 'top',
      panelClass: 'snackbar-ok',
      duration: 5000
    })
  }
}
```