laSalle
UNIVERSITAT RAMON LLULL

Disseny i Programació Orientada a Objectes

Pràctica S1 – Documentació de l'API

Centre Comercial el Cofre

Escola Tècnica Superior d'Enginyeria
La Salle - Universitat Ramon Llull

ÍNDEX

1 - INTRODUCCIÓ

Internet ha esdevingut una eina omnipresent en la nostra societat. Des del primer missatge que enviem per Discord al llevar-nos fins l'entrega que fem a l'eStudy just abans d'anar a dormir, la nostra vida s'ordena al voltant d'aquest sistema de comunicació.

Concretament, la majoria de plataformes amb les que interaccionem fan servir el protocol HTTP per a comunicar les aplicacions dels nostres dispositius amb els corresponents servidors. Com que a l'assignatura de POO no volem ser menys, també hem decidit fet servir aquest protocol per comunicar-nos amb l'API del sistema de compres digital que estem implementant.

En aquest document trobareu una introducció al protocol HTTP, així com una especificació en detall de l'API del sistema, que ja ha estat dissenyada i desenvolupada per l'equip de l'assignatura. Finalment es presenten diferents formes de treballar amb aquest protocol en Java.

2 - PROTOCOL HTTP

El protocol HTTP és un dels fonaments del món web modern. Quan accedim a una pàgina web amb el nostre navegador preferit estem fent peticions HTTP, tot i que sigui transparent per nosaltres. De forma similar, la gran majoria de les peticions que fan les aplicacions dels nostres dispositius mòbils també segueixen aquest protocol.

Concretament, es tracta d'un protocol d'aplicació (capa 7 de la torre OSI) de comunicació client-servidor. Aquesta direccionalitat és important, ja que tota la comunicació es realitzarà en forma de peticions puntuals del client al servidor, que hi donarà resposta de forma síncrona.

Si bé el protocol ha anat evolucionant al llarg del temps, incorporant tècniques més avançades, la idea general del seu funcionament segueix sent la que es pot apreciar a la Figura 1.

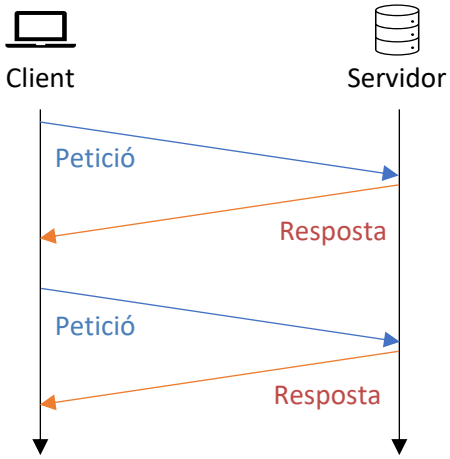

Figura 1. Comunicació entre client i servidor en el protocol HTTP.

A continuació s'explicarà el funcionament de les peticions i les respostes en aquest protocol.

2.1 - Peticions

Les peticions HTTP són trames que envia el client al servidor. Estan formades per quatre parts destacables, entre d'altres:

- ***Method*:** El mètode, una paraula que representa l'objectiu de la petició. Els més habituals són GET (per obtenir recursos del servidor) i POST (per crear-ne un de nou), però també n'hi ha d'altres com DELETE (per eliminar-ne un d'existent). Si voleu més informació, consulteu: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods
- ***Path*:** El camí del recurs sobre el que aplica la petició. Tot i que la trama el simplifica, correspon a un URL del servidor. Al final d'aquest capítol es detalla aquest punt.
- ***Headers*:** Capçaleres que pot incloure (o no) el client per donar informació extra al servidor sobre la seva petició, com el format del *body*. Si voleu més informació, consulteu el recurs: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers
- ***Body*:** El cos de la petició, que pot estar buit. Habitualment es tracta de text en un format conegut tant pel client com el servidor. Avui en dia és força comú fer servir el format JSON. El client sol enviar

informació només en aquells mètodes que ho requereixen (per exemple, la informació del recurs a crear en un POST).

2.2 - Respostes

Una resposta HTTP és la trama que envia el servidor al client després de rebre una petició. Estan formades per tres parts destacables, entre d'altres:

- ***Status code*:** Enter que indica l'estat de la resposta. En general, indica si s'ha pogut satisfer la petició correctament. Si voleu més informació, consulteu el recurs: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
- ***Headers*:** Capçaleres que pot incloure (o no) el servidor per donar-nos informació extra sobre la seva configuració, el format de la resposta o altres. Si voleu més informació, consulteu el recurs: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers
- ***Body*:** El cos de la resposta, que pot estar buit. Habitualment es tracta de text en un format conegut tant pel servidor com el client. Avui en dia és força comú fer servir el format JSON.

2.3 - Format dels URL

Com s'ha explicat, les peticions HTTP inclouen el *path* sobre el que s'estan realitzant. Conceptualment aquest *path* correspon a un URL del servidor, que segueix un format estàndard. A la Figura 2 se'n pot apreciar un exemple.

```
https://api.server.com/endpoint/path?id=2&name=example&public=true
```

Figura 2. Exemple d'un URL gairebé complet.

Tot i que la sintaxi dels URL permet afegir-hi més elements, aquest exemple visualitza les parts més rellevants pel protocol HTTP, sobretot pel que fa a APIs. A continuació s'expliquen per separat:

- ***Protocol* (`https://`):** Habitualment serà *http* o *https*, però és bo tenir present que en altres contextos es fan servir altres protocols. Recordeu que en el món web és preferible treballar amb *https* per evitar la intercepció de les nostres dades.
- ***Host* (`api.server.com`):** El nom de domini del servidor amb el que ens comuniquem (amb les seves diferents parts, com el subdomini).
- ***Endpoint* (`/endpoint/path`):** La ubicació (*path*) dins el servidor a la que estem enviant la petició. Es tracta d'un seguit de segments separats pel caràcter */*. En el següent capítol es descriuen els *endpoints* disponibles a l'API que heu de fer servir, així com el seu funcionament.
- ***Query* (`?id=2&name=example&public=true`):** De forma opcional, un URL pot contenir informació extra per al servidor. Concretament, es poden afegir tants paràmetres com vulguem a partir del caràcter *?*. Cada paràmetre s'especifica amb el seu nom, seguit d'un = i el seu valor (que pot ser del tipus que calgui). Els paràmetres es separen amb el caràcter *&*, i el seu ordre és indiferent.

Tingueu present que en un URL no hi poden haver caràcters especials, pel que cal realitzar el procés conegut com a *URL encoding*. Podeu trobar-ne més informació al recurs: https://developer.mozilla.org/en-US/docs/Glossary/percent-encoding

3 - API DEL SISTEMA EL COFRE

A l'etapa final de la pràctica de l'assignatura se us demana comunicar el vostre programa (client) amb l'API dissenyada per l'equip de professors (servidor). L'URL base d'aquesta API es pot trobar a la Figura 3. Si hi accediu des del navegador, podreu veure'n la documentació en un format lleugerament diferent al d'aquest document.

```
https://balandrau.salle.url.edu/dpoo
```

Figura 3. URL base de l'API del sistema el Cofre.

Tingueu present que només podreu accedir al servidor *balandrau* si esteu connectats a la xarxa *eduroam* del campus, o si us connecteu a la VPN de la universitat. Podeu trobar més informació de com fer-ho al document corresponent de l'eStudy.

Per a implementar els requisits exposats en l'enunciat corresponent, disposeu d'un conjunt d'*endpoints*, que s'expliquen a continuació.

3.1 - Productes

Per gestionar els vostres productes, caldrà que proporcioneu el vostre identificador de grup com a part de l'URL de l'*endpoint* que estigueu fent servir. Al llarg del document es farà referència a aquest valor amb la notació **{id}**. L'identificador de grup correspon al valor que vau escollir a l'eina de l'eStudy, per exemple **P1-G42**.

3.1.1 - Crear producte

Per crear un producte nou, caldrà realitzar una petició **POST** a l'*endpoint* **/{id}/products**. El *body* d'aquesta petició ha de contenir les dades del producte en el format JSON especificat a l'enunciat corresponent.

La Figura 4 mostra exemple d'aquest *endpoint*, incloent el *body*. També s'inclou el valor correcte que caldria proporcionar al *header* "Content-Type", tot i que aquesta API no el requereix.

```
POST https://balandrau.salle.url.edu/dpoo/P1-G42/products
Headers: Content-Type="application/json"
Body:  { "name": "Ibuprofen Capsules", "brand": "Generic Pharma",
         "mrp": 5.5, "category": "SUPER_REDUCED", "reviews": [] }
```

Figura 4. Exemple de l'endpoint per crear un producte, incloent els headers (opcionals) i el body (obligatori).

Tingueu present que l'API no comprova lògica de negoci, només valida que el format JSON del *body* sigui correcte.

3.1.2 - Consultar productes

L'*endpoint* **/{id}/products** respon a peticions **GET** amb la vostra llista de productes. El cos de la resposta segueix el format JSON utilitzat en la seva creació.

Aquest *endpoint* accepta paràmetres en forma de *query* per filtrar els productes segons els seus atributs. Si l'atribut és un String s'aplicarà lògica *case-insensitive* i d'igualtat parcial (*contains*). En cas que hi hagi més d'un paràmetre, es retornarà aquells productes que satisfacin totes les condicions (com si es tractés d'una operació *and*).

Addicionalment, es disposa d'un *sub-endpoint* per obtenir el producte d'una posició concreta de la llista (començant per 0). Si la posició no existeix, retorna un error en format JSON.

La Figura 5 mostra les tres casuístiques mencionades amb tres URLs diferents. Sentiu-vos lliures de provar-los al vostre navegador per veure'n els resultats, però tingueu present que per rebre valors primer haureu d'haver creat un producte com a mínim.

```
GET https://balandrau.salle.url.edu/dpoo/P1-G42/products
GET https://balandrau.salle.url.edu/dpoo/P1-G42/products?name=xuixo
GET https://balandrau.salle.url.edu/dpoo/P1-G42/products/2
```

Figura 5. Exemples d'URLs per l'endpoint de consulta de productes, amb els mètodes corresponents.

3.1.3 - Eliminar productes

L'eliminació de productes funcionarà de forma similar a la consulta, però acceptant peticions **DELETE** a l'*endpoint* corresponent.

L'URL amb paràmetres *query* eliminarà els productes que satisfacin totes les condicions establertes, però en aquest cas la comprovació serà exacta (*case-sensitive* i *equals*). Si no hi ha cap producte que satisfaci les condicions establertes es retornarà un error en format JSON. De forma similar, es retornarà un error si no es proporciona cap paràmetre.

També es podran eliminar productes segons la seva posició a la llista (començant per 0).

A la Figura 6 s'exemplifiquen les dues casuístiques mencionades amb dos URLs diferents.

```
DELETE https://balandrau.salle.url.edu/dpoo/P1-G42/products?name=xuixo
DELETE https://balandrau.salle.url.edu/dpoo/P1-G42/products/3
```

Figura 6. Exemples d'URLs per l'endpoint d'eliminació de productes, amb els mètodes corresponents.

3.2 - Botigues

Els *endpoints* per crear, consultar i eliminar botigues són equivalents als de productes. La Figura 7 llista tots els URLs corresponents, en el mateix ordre que s'han detallat a l'apartat anterior.

```
POST https://balandrau.salle.url.edu/dpoo/P1-G42/shops
Headers: Content-Type="application/json"
Body:  { ... }

GET https://balandrau.salle.url.edu/dpoo/P1-G42/shops
GET https://balandrau.salle.url.edu/dpoo/P1-G42/shops?name=vagrant
GET https://balandrau.salle.url.edu/dpoo/P1-G42/shops/0

DELETE https://balandrau.salle.url.edu/dpoo/P1-G42/shops?name=queviures
DELETE https://balandrau.salle.url.edu/dpoo/P1-G42/shops/3
```

Figura 7. Exemples d'URLs pels endpoints de gestió de botigues, amb els mètodes corresponents.

Tingueu present que l'API no comprova lògica de negoci, només valida que el format JSON del *body* sigui correcte.

4 - PETICIONS HTTP EN JAVA

En aquest apartat s'expliquen dues formes de realitzar peticions HTTP en Java. La primera consisteix a fer ús d'una llibreria auxiliar proporcionada per l'equip de l'assignatura, mentre que la segona fa ús de les eines incloses a Java per defecte. Sentiu-vos lliures de fer servir l'enfoc que us agradi més.

4.1 - Llibreria ApiHelper

Per facilitar-vos la interacció amb l'API del sistema elCofre, se us proporciona la llibreria auxiliar *ApiHelper*, que disposa de mètodes per abstraure les peticions GET, POST i DELETE. També se us proporciona el JavaDoc, detallant-ne el funcionament mètode a mètode. A la Figura 8 podeu observar-ne el diagrama de classes UML.

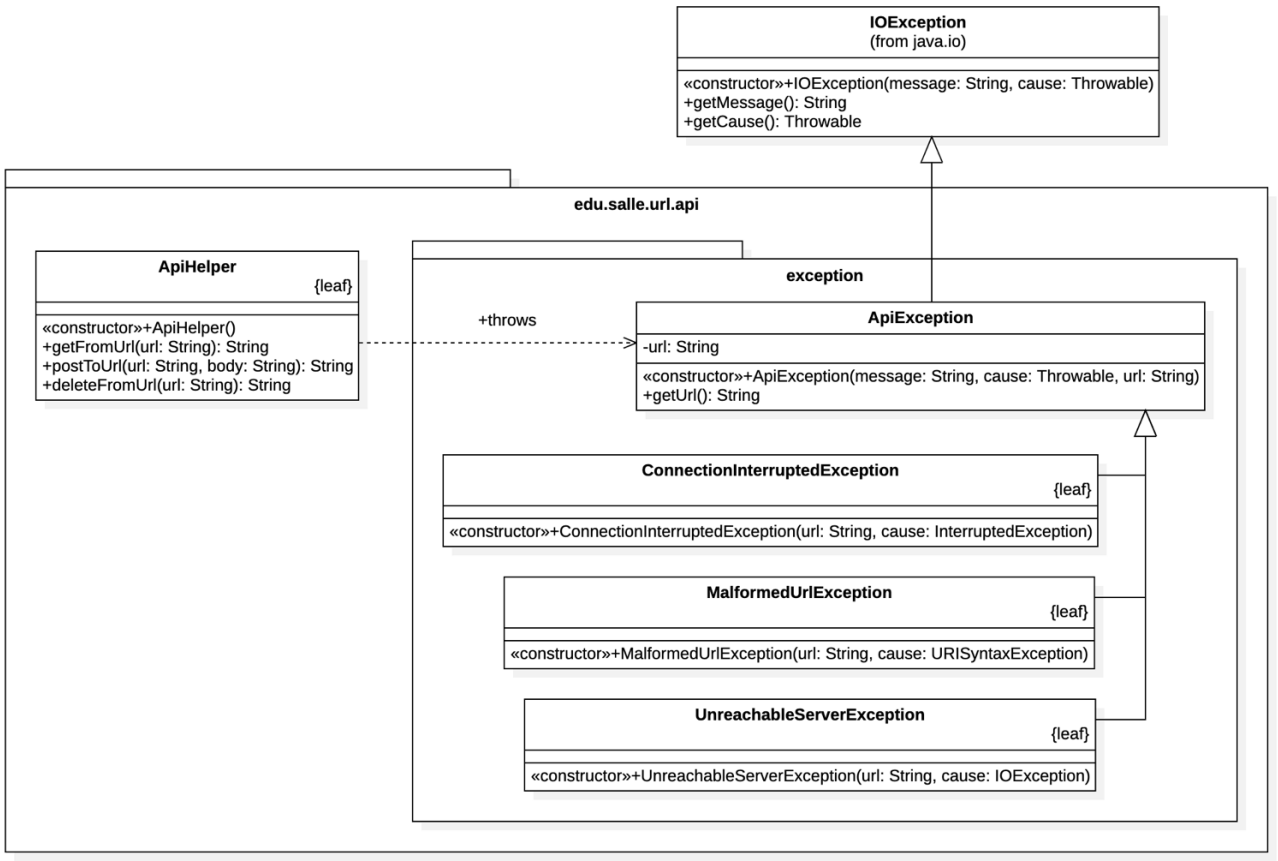

Figura 8. Diagrama de classes UML de la classe ApiHelper.

ApiHelper és la principal classe amb la que interaccionareu. Tanmateix, és important tenir present que els seus mètodes poden llençar excepcions pròpies per diferents motius. Tot i que us recomanem gestionar els errors amb la classe general **ApiException**, podeu fer servir les seves subclasses per a distingir entre causes concretes:

- **ConnectionInterruptedException:** Generada quan s'interromp la connexió amb l'API durant una crida.
- **MalformedUrlException:** Generada quan es detecta un error de sintaxi en l'URL al que us voleu connectar.
- **UnreachableServerException:** Generada quan l'ordinador no pot connectar-se al servidor de l'API. Pot ser que el servidor estigui caigut o que no estigueu connectats a la VPN.

A continuació trobareu un resum del funcionament dels mètodes de la classe **ApiHelper**.

- **ApiHelper:** El constructor de la classe. S'encarrega de comprovar automàticament si es pot establir una connexió a l'API, llençant una ApiException si hi ha cap problema. Recordeu que en aquesta situació cal fer servir els fitxers JSON locals. Si us trobeu amb un error inesperat, contacteu amb l'equip de l'assignatura.
- **getFromUrl:** Mètode per fer una petició GET a l'API. Rep l'URL a consultar i en retorna els continguts. Pot llençar una ApiException en els casos comentats anteriorment.
- **postToUrl:** Mètode per fer una petició POST a l'API. Rep l'URL on fer-la i el contingut a enviar. Retorna la resposta del servidor, en cas que n'hi hagi. Pot llençar una ApiException en els casos comentats anteriorment.
- **deleteFromUrl:** Mètode per fer una petició DELETE a l'API. Rep l'URL on fer-la i retorna la resposta del servidor, en cas que n'hi hagi. Pot llençar una ApiException en els casos comentats anteriorment.

4.2 - Eines HTTP en Java

Aquest apartat descriu eines natives per a realitzar peticions HTTP en Java, com a alternativa a la llibreria ApiHelper.

Inicialment, per a implementar un client en Java que realitzés peticions HTTP calia fer ús de classes de propòsit força general com *URLConnection* o *BufferedReader*. Si bé aquesta forma de treballar segueix sent vàlida, és força manual i sovint no acaba de representar fidelment el que volem fer.

Per solucionar aquests problemes, Java 11 va introduir un conjunt de classes de propòsit més específic com *HttpClient* o *HttpRequest*. L'objectiu d'aquestes noves classes és oferir una forma senzilla de treballar amb HTTP com a protocol de comunicació, abstraient-nos de les seves particularitats de "baix nivell".

En els següents apartats es presenten exemples de com fer peticions GET i POST amb cadascun d'aquests conjunts d'eines. Podeu investigar com es realitzaria una petició DELETE.

4.2.1 - Eines de propòsit general

La Figura 9 presenta un exemple de petició GET amb les eines més antigues (de propòsit general). A la Figura 10 es pot observar un exemple de petició POST.

```
private String getStringFromUrl(String url) {
    try {
        // Define the URL, its corresponding connection and set up
        // a BufferedReader to read from it
        URL auxUrl = new URL(url);
        URLConnection con = auxUrl.openConnection();
        InputStreamReader aux = new InputStreamReader(con.getInputStream());
        BufferedReader in = new BufferedReader(aux);

        // Consume the reader
        StringBuilder stringBuilder = new StringBuilder();
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            stringBuilder.append(inputLine);
        }

        // Close the reader
        in.close();
        return stringBuilder.toString();
    } catch (IOException e) {
        // Note: This IOException encompasses a possible MalformedURLException
        // TODO manage properly.
        System.err.println(e.getMessage());
        return "";
    }
}
```

Figura 9. Exemple de lectura dels continguts d'un URL amb eines de propòsit general en Java, equivalent a una petició GET.

```java
private void postStringToUrl(String url, String body) {
    try {
        // Define the URL and its corresponding HTTP connection, set up
        // the method and establish that we want to output data as well
        // as to have specific headers (properties)
        URL auxUrl = new URL(url);
        HttpURLConnection con = (HttpURLConnection) auxUrl.openConnection();
        con.setDoOutput(true);
        con.setRequestMethod("POST");
        con.setRequestProperty("Content-Type", "application/json");

        // Set up an OutputStreamWriter to send the body to the connection
        OutputStreamWriter out = new OutputStreamWriter(con.getOutputStream());
        out.write(body);
        out.close();

        // Set up and consume a BufferedReader to read from the connection, in
        // case the API returns something (we could also choose to ignore it)
        InputStreamReader aux = new InputStreamReader(con.getInputStream());
        BufferedReader in = new BufferedReader(aux);

        StringBuilder stringBuilder = new StringBuilder();
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            stringBuilder.append(inputLine);
        }

        // Close the reader
        in.close();
        System.out.println(stringBuilder);
    } catch (IOException e) {
        // Note: This IOException encompasses a possible MalformedURLException
        // TODO manage properly.
        System.err.println(e.getMessage());
    }
}
```

Figura 10. Exemple d'escriptura de contingut a un URL amb eines de propòsit general en Java, equivalent a una petició POST.

4.2.2 - Eines de propòsit específic

La Figura 11 presenta un exemple de petició GET amb les eines més modernes (de propòsit específic). A la Figura 12 es pot observar un exemple de petició POST.

```
private String getStringFromUrl(String url) {
    try {
        // Define the request
        // The default method is GET, so we don't need to specify it
        // (but we could do so by calling .GET() before .build()
        // The HttpRequest.Builder pattern offers a ton of customization for
        // the request (headers, body, HTTP version...)
        HttpRequest request = HttpRequest.newBuilder()
                .uri(new URI(url))
                .build();

        // We have to create a client to send the request
        // This could also be done by using the HttpClient.Builder pattern
        // if we needed to customize it, but we don't
        HttpClient client = HttpClient.newHttpClient();

        // We use the default BodyHandler for strings
        // (so we can get the body of the response as a String)
        // Note we could also send the request asynchronously, but things
        // would escalate in terms of coding complexity
        HttpResponse<String> response = client.send(
                request,
                HttpResponse.BodyHandlers.ofString()
        );

        // Just return the body
        return response.body();
    } catch (URISyntaxException | IOException | InterruptedException e) {
        // TODO manage properly.
        System.err.println(e.getMessage());
        return "";
    }
}
```

Figura 11. Exemple de petició GET amb eines de propòsit específic en Java.

```java
private void postStringToUrl(String url, String body) {
    try {
        // Define the request
        // In this case, we have to use the .POST() and .headers() methods
        // to define what we want (to send a string containing JSON data)
        HttpRequest request = HttpRequest.newBuilder()
                .uri(new URI(url))
                .headers("Content-Type", "application/json")
                .POST(HttpRequest.BodyPublishers.ofString(body))
                .build();

        // We have to create a client to send the request
        // This could also be done by using the HttpClient.Builder pattern
        // if we needed to customize it, but we don't
        HttpClient client = HttpClient.newHttpClient();

        // We could use a BodyHandler that discards the response header, but
        // here we print the API's response (and could process it if needed)
        // Note we could also send the request asynchronously, but things
        // would escalate in terms of coding complexity
        HttpResponse<String> response = client.send(
                request,
                HttpResponse.BodyHandlers.ofString()
        );
        System.out.println(response.body());

        // Alternative we could use to discard the response:
        // client.send(request, HttpResponse.BodyHandlers.discarding());

    } catch (URISyntaxException | IOException | InterruptedException e) {
        // TODO manage properly.
        System.err.println(e.getMessage());
    }
}
```

Figura 12. Exemple de petició POST amb eines de propòsit específic en Java.