

Pràctica 1

API REST PER UNA PÀGINA WEB DE ARTICLES

Sistemes Oberts

KEVIN SÁNCHEZ RAMÍREZ
IULIAN SEBASTIAN OPREA

Índex

Índex	1
1. Introducció	2
2. Estructura de la Pràctica	2
3. Decisions de disseny	7
4. Jocs de proves realitzats.....	9
5. Conclusions.....	27
6. Manual d'instal·lació	28
7. Video	30

1. Introducció

Aquesta pràctica numero 1 correspon al disseny del que anomenen avui en dia “BACKEND” de una pagina web que tracta sobre articles de tecnologia. Aquest part esta definida sent una API REST, on treballarà conjuntament amb el JPA ,extrauran informació de la taula de bases de dades corresponent per poder mostrar-la al usuari.

Aquestes peticions es realitzarà en contingut JSON o XML segons la preferència del usuari i es tornarà amb el tipus que el usuari prefereixi. A part d'això les diferents classes que estan preparades per a interactuar amb el usuari estan preparades per a poder agafar qualsevol informació de qualsevol taula fent així una pàgina web que valida totes les necessitats de un usuari registrar i no registrat amb les seves restriccions pertinents.

2. Estructura de la Pràctica

La practica està distribuïda en tres diferents packages:

- Package “**authn**” la qual conte el següents fitxers:
 - **Credentials** : conté la informació necessaria per a poder mantenir les credencials de qualsevol client
 - **RestRequestFilter** : valida les credencials introduïdes per l'usuari
 - **Secured** : anotació que aplicara el RestRequestFilter
- Package “**model.entities**”
 - **Article** : conté la informació que fa falta tenir sobre qualsevol article
 - **ArticleDTO** : classe que ens fara com a interfície (Data Transfer Output) per a que nomes mostressiem la informació de la segona figura
 - **ArticleGQ** : classe que ens fara com a interfície (Data Transfer Output) per a que nomes mostressiem la informació de la primera figura
 - **ArticleIN**: classe que ens serveix per poder agafar la informació que el article necessita per ser creat, això seria un Data Transfer Input.
 - **Topics** : enumeració del diferents topics que pot tindre un article

- **Customer:** conté la informació necessària per mantindre un estat d'aquest client
 - **CustomerDTO:** classe auxiliar (Data Transfer Output) que ens serveix per mostrar la informació de forma reduïda en llistat general
 - **Links:** classe que podrà mantindre el format corresponent a l'especificat dels links, aquesta només ens guarda l'hiperenllaç a l'últim article publicat (principi **HATEOAS**). D'aquesta forma la nostra sortida tindrà una estructura "links -> link: /article/id".
-
- Package "**service**"
 - **RestApp** : conte la ruta que ha de seguir la nostra pàgina web (rest/api/v1)
 - **ArticleFacadeRest** : classe que implementa les API REST de article
 - **CustomerRest** : classe que implementa les funcionalitats del customer, aquest combina les interaccions del JPA amb les del API REST
 - **AbstractFacade:** Implementa les funcions bàsiques que hauria de fer qualsevol unitat de persistència amb la base de dades

Hi ha 4 entitats de les quals 3 estan creades per nosaltres i 1 per el mestre Marc Sánchez específicament la taula Credencials. Les nostres son les següents:

Pel que fa a l'**article**, s'ha emmagatzemat:

Identificador del article : Long

La privacitat del article : Booleà

El títol del article : String

El resum del article : String màxim de 20 caràcters

El text del article: String de màxim 500 caràcters

Visualitzacions del article: Integer

Data de la creació del article: LocalDate

Tòpic del article: TopicTaula

Autor: Customer (Relació ManyToOne)

Pel que fa la taula **Article_Topics** , s'ha emmagatzemat:

El identificar del article: Long

El tòpic : Tòpic (Web_Programming, Python, HMTL, CSS, JavaScript)

I per últim la taula **customer**, s'ha emmagatzemat:

Identificador del customer: Long

Data de creació: LocalDate

Descripció: String

Username : String (inicialment sense valor, s'assigna al fer lo primer get)

Credencials: Long (clau forana a la taula credencials)

Els serveis REST que s'han implementat són els següents (el que demana el enunciat de la practica, el obligatoris més el opcionals):

Mètode HTTP	Crida	Descripció
GET	<code>/rest/api/v1/article?topic=\${topic}&author=\${author}</code>	Retorna una llista amb elements de la figura 1, potser JSON o XML, amb elements que compleixin les condicions especificades.
GET	<code>/rest/api/v1/article/\${id}</code>	Retorna l'article amb forma de la figura 2 i augmenta el numero de visualitzacions
DELETE	<code>/rest/api/v1/article/\${id}</code>	Borra el article especificat per el paràmetre id
POST	<code>/rest/api/v1/article</code>	Crea un nou article fent les pertinents validacions i ficant la data del dia que es crea aquest article.
GET	<code>/rest/api/v1/customer</code>	Retorna una llista de customers on s'indica si és només lector o és autor(ha escrit mínim un article), també es retorna un enllaç a l'últim article publicat per aquest usuari, en cas que sigui autor. I informació breu com lo seu nom d'usuari i foto de perfil
GET	<code>/rest/api/v1/customer/\${id}</code>	Retorna tota la informació que tenim guarda d'aquest usuari en la nostra base de dades (menys informació confidencial, com és la contrasenya), el format d'aquest canvia amb el GET anterior, ja que en aquest cas tindrem informació més especifica, com quins articles ha publicat i informació d'aquest, la data de creació de l'usuari,

		una breu descripció a més de tot lo que es mostra en el llistat general
PUT	/rest/api/v1/customer/\${id}	Permet modificar les dades d'un customer mitjançant la seva id, aquest ha de modificar el customer amb id que tingui les credencials que s'han introduït, ja que d'altra manera mostrarà un error de falta de permisos, ens permet modificar el perfil del customer, canviant el username (afecta també a les credencials que haurà d'introduir), la descripció i la foto de perfil

3. Decisions de disseny

Per tal de realitzar la pràctica hem utilitzat el projecte base del moodle i hem anat generant el nostre codi.

Les decisions de disseny que hem fet per tot el que te a veure amb **Article** ha sigut:

Crear dos classes addicionals anomenades **ArticleDTO** i **ArticleGQ** que actuen como una capa per a que nomes es vegi nomes lo que la figura especifica vol que es vegi.

Tenim que tindre dos tòpics llavors per a que en **ArticleService** després sigui més fàcil buscar si coincideix amb aquest tòpic donant igual en quina posició de la taula esta s'ha ficat en una llista però també s'ha ficat certes anotacions com ara ve a ser **@ElementCollection (targetClass = Topic.class)** y **@Enumerated(EnumType.STRING)** que ens dona la possibilitat de comprovar el problema que em especificat amb anterioritat.

Per la funció **llistatArticles** que ve donat per el GET **/rest/api/v1 /article?topic = \${topic}&author=\${author}** enlloc de tindre 9 casos el que hem fet es que es crearà una query especifica per a cada cas segons els arguments que ens passi la classe ArticleRest llavors ens quedaria un codi menys reduït perquè no tindríem que crear un cas per cada cas sinó que anem creant la query sobreposant sobre aquella base tot lo necessari.

L'ultima decisió de disseny y una de les més importants donat que afecta a la segona decisió de disseny es que Tòpic ha de ser una enumeració llavors no podem modificar la enumeració el qual ens afavoreix en l'aspecte de no se puguin afegir de nous encara que el usuaris i vulguin, sols poden triar en el que ja hi ha.

D'altra banda les decisions de disseny respecte al **Customer** són les següents:

La classe **Customer** serà la classe que contindrà tota la informació que tenim dels usuaris en la nostra base de dades, aquesta informació serà mostrada quan féssim una cerca directa sobre aquest (GET Customer per ID). A més que s'ha decidit fer un mètode copia per tal de poder eliminar la recursivitat que hi ha entre customers i articles. A més, s'afegit una nova relació OneToOne de customers i

credentials, ja que cada credencials correspondrà a un sol usuari, això ens permet tindre un millor control de l'estat dels comptes, permetint les modificacions de les dades de l'usuari només si són d'ell. Lo link de l'usuari s'actualitzarà només en el moment de fer el get, per tindre l'últim de la llista en tot moment, d'aquesta forma cridem al mètode toString de l'article, que hem sobreescrit per a que ens retorni el format següent "/article/id". També, s'ha fet un mètode que elimina articles de la llista i actualitza el link si es que ha esborrat l'últim article, en cas contrari només l'esborra de la llista.

S'ha creat una classe Links per mantindre la estructura que es demana sobre aquests. Només lo mostrarem en el cas que l'usuari sigui autor, això ho farem mitjançant una condició la qual retorna null si no és escriptor de algun article, això permet que no serialitze el camp.

També hem creat una classe **CustomerDTO** la qual ens permet mostrar informació menys detallada del nostre usuari, això ens permet que els mètodes de l'api Rest s'adaptin millor, ja que quan mostrem el llistat de tots els usuaris, no ens interessa tant obtindre dades detallades, com si ho faríem en el moment que hi accedim al seu perfil.

4. Jocs de proves realitzats

Per tal de veure que realment el treball que hem fet esta ben implementat segons les nostres decisions de disseny i que implementa be els diferents mètodes que ens demana aquesta practica ho hem fet amb Postman, on ajuntem totes la collecció a continuació en aquest enllaç, però també expliquem que fa cada cas.

Enllaç Postman amb les diferents proves:

<https://sob-practica.postman.co/workspace/SOB-Practica-Workspace~2cbb3e2a-cb7c-411a-ace4-9dd2db930f49/collection/39916940-63a4f6e0-e3c5-463d-84b3-7d44fb13ba89?action=share&creator=39916940>

Taula que descriu el diferents casos dels diferents testos en Postman:

Cas 1

GET /rest/api/v1/article?topic=\${topic}

Validar que ens mostri la informació de la figura numero 1 i que el resultat nomes sigui d'aquells articles que tenen aquell tòpic

Sortida esperada : Codi 200 OK , la Informació en XML ,seguint la forma de la figura numero 1 i mostrant nomes informació dels articles que tenen aquell tòpic.

Sortida real:

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <articleGQs>
3      <articleGQ>
4          <autor>sob</autor>
5          <data>2024-11-12</data>
6          <imatge>C:\Iulian\image.png</imatge>
7          <resum>Resum de 20 paraules</resum>
8          <titol>NomArticle</titol>
9          <visualitzacions>5000</visualitzacions>
10     </articleGQ>
11 </articleGQs>
```

Correcte : SI

Cas 2

GET /rest/api/v1/article?author=\${author}

Validar que ens mostri la informació de la figura numero 1 i que el resultat només sigui d'aquells articles que tenen aquell autor especificat per QueryParam.

Sortida esperada : Codi 200 OK , la Informació en JSON ,seguint la forma de la figura numero 1 i mostrant només informació dels articles que tenen aquell autor.

Sortida real:

```
1  [
2      {
3          "autor": "kevin2004",
4          "data": "2024-01-22",
5          "imatge": "C:\\Users\\iulia\\Downloads\\imatge.png",
6          "resum": "Resum ART 2",
7          "titol": "Article 2",
8          "visualitzacions": 300
9      }
10 ]
```

Correcte : SI

Cas 3

GET /rest/api/v1/article?topic=\${topic}&author=\${author}

Validar que ens mostri la informació de la figura numero 1 i que la nostra api pot admetre per paràmetre un tòpic i un autor.

Sortida esperada: Codi 200 OK , la Informació en XML, seguint la forma de la figura numero 1 i mostrant només informació dels articles que tenen aquell autor i un dels dos tòpics possibles que sigui un del especificat per QueryParam.

Sortida real:

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <articleGQs>
3      <articleGq>
4          <autor>kevin2004</autor>
5          <data>2024-01-22</data>
6          <imatge>C:\\Users\\iulia\\Downloads\\imatge.png</imatge>
7          <resum>Resum ART 2</resum>
8          <titol>Article 2</titol>
9          <visualitzacions>300</visualitzacions>
10     </articleGq>
11 </articleGQs>
```

Correcte : SI

Cas 4

GET /rest/api/v1/article?topic=\${topic}&topic=\${topic}&author=\${author}

Validar que ens mostri la informació de la figura numero 1 i que la nostra api pot admetre per paràmetre dos tòpics i un autor.

Sortida Esperada:

Codi 200 OK , la Informació en JSON, seguint la forma de la figura numero 1 i mostrant nomes informació dels articles que tenen aquell autor i exactament aquests dos tòpics sense importar la seva informació.

Sortida Real:

```
1  [
2      {
3          "autor": "kevin2004",
4          "data": "2024-01-22",
5          "imatge": "C:\\Users\\iulia\\Downloads\\imatge.png",
6          "resum": "Resum ART 2",
7          "titol": "Article 2",
8          "visualitzacions": 300
9      }
10 ]
```

Correcte : SI

Cas 5

GET /rest/api/v1/article

Mostrar nomes la informació de la figura 1 i que ens retorna tots els articles que hi tenim a la base de dades

Sortida Esperada: Codi 200 OK i la informació en format XML.

Sortida Real:

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <articleGQs>
3      <articleGq>
4          <autor>sob</autor>
5          <data>2024-11-12</data>
6          <imatge>C:\\Iulian\\image.png</imatge>
7          <resum>Resum de 20 paraules</resum>
8          <titol>NomArticle</titol>
9          <visualitzacions>5000</visualitzacions>
10     </articleGq>
11     <articleGq>
12         <autor>kevin2004</autor>
13         <data>2024-01-22</data>
14         <imatge>C:\\Users\\iulia\\Downloads\\imatge.png</imatge>
15         <resum>Resum ART 2</resum>
16         <titol>Article 2</titol>
17         <visualitzacions>300</visualitzacions>
18     </articleGq>
19 </articleGQs>
```

Correcte : SI

Cas 6

GET /rest/api/v1/article/\${id}

Volem veure la informació de la figura 2 i que podem veure el article si es públic encara de no estar registrat i que també augmenta el nombre de visualitzacions

Sortida Esperada: Codi 200 OK, informació en format JSON, que hem obtingut el article desitjat i que el numero de visualitzacions a augmentat en 1.

Sortida Real:

```
1 {
2   "autor": "sob",
3   "data": "2024-11-12",
4   "imatge": "C:\\Iulian\\image.png",
5   "text": "Tenim el següent text que es de prova i nomes per veure que install jsp funciona a la perfeccio",
6   "titol": "NomArticle",
7   "visualitzacions": 5001
8 }
```

Correcte : SI

Cas 7

GET /rest/api/v1/article/\${id}

Volem veure la informació de la figura 2 i que no podem veure el article si es privat perquè no estem registrat i no augmenta el numero de visualitzacions

Sortida Esperada: El numero de visualitzacions es queda igual y en retorna el codi de error 403 fent referencia ha que no estem autoritzats perquè hem introduït alguna cosa malament o no estem registrats.

Sortida Real:

```
Body Cookies Headers (5) Test Results (1/1) 403 Forbidden 14 ms 1.25 KB Save Response
Pretty Raw Preview Visualize HTML
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"><html xmlns="http://www.w3.org/1999/xhtml"><head><title>Eclipse GlassFish 6.2.1 - Error report</title><style type="text/css"><!--H1 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:22px;} H2 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:16px;} H3 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:14px;} BODY {font-family:Tahoma,Arial,sans-serif;color:black;background-color:white;} B {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;} P {font-family:Tahoma,Arial,sans-serif;background:white;color:black;font-size:12px;}A {color : black;}HR {color : #525D76;}--></style></head><body><h1>HTTP Status 403 - Forbidden</h1><hr/><p><b>Status report</b><p><b>message</b>Forbidden</p><p><b>description</b>Access to the specified resource has been forbidden.</p></body></html>
```

Correcte : SI

Cas 8

GET /rest/api/v1/article/\${id}

Volem veure la informació de la figura 2 i que podem veure el article si es privat perquè estem registrat i augmenta el numero de visualitzacions

Sortida Esperada: Codi 200 OK, informació en format JSON, que hem obtingut el article desitjat i que el numero de visualitzacions a augmentat en 1.

Sortida Real:



Correcte : SI

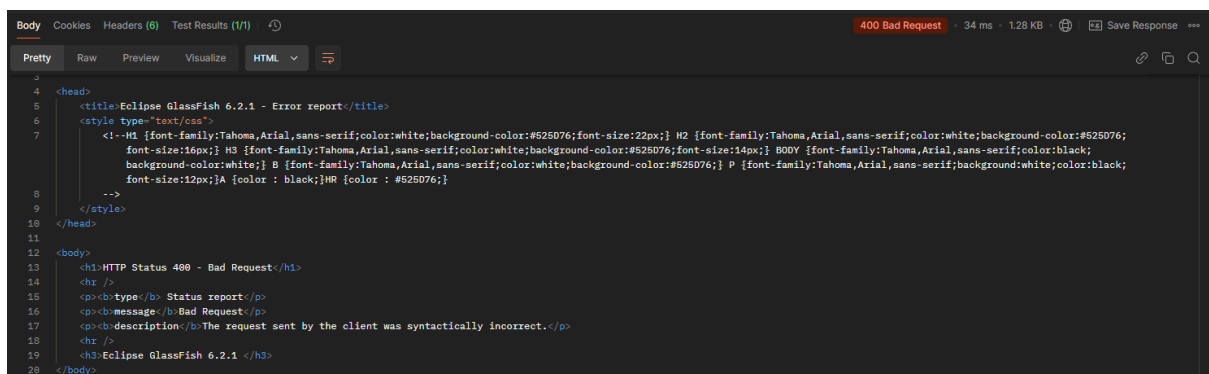
Cas 9

POST /rest/api/v1/article

Volem comprova que nomes insereix el article si aquest es vàlid

Sortida Esperada: Codi 400 fent referencia a “BAD_REQUEST” perquè hi ha alguna cosa que no hi es valida.

Sortida Real:



Correcte : SI

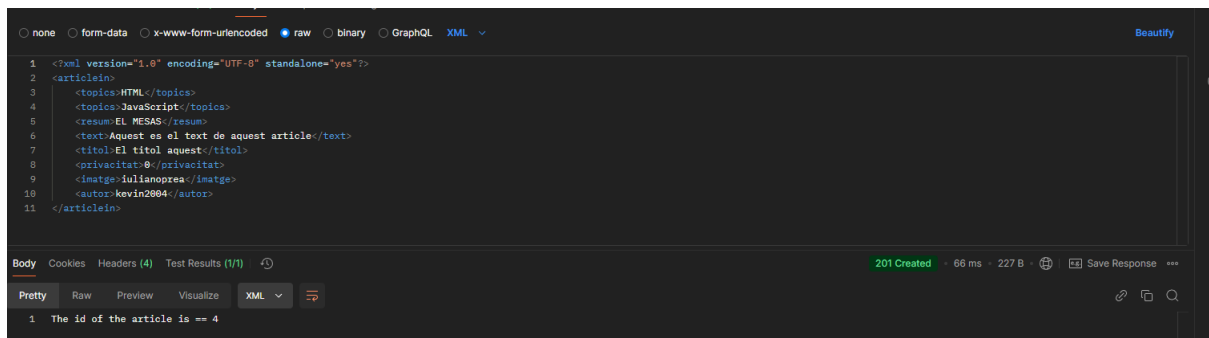
Cas 10

POST /rest/api/v1/article

Volem comprova que insereix el article perquè es vàlid

Sortida Esperada: Codi 201 fent referencia a “Created” perquè s’ha creat el nou article amb la informació desitjada.

Sortida Real:



The screenshot shows a REST client interface with the following details:

- Request Body (XML):**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<articlein>
  <topics>HTML</topics>
  <topics>JavaScript</topics>
  <resum>EL MESAS</resum>
  <text>Aquest es el text de aquest article</text>
  <titol>El titol aquest</titol>
  <privacitat>0</privacitat>
  <imatge>julianoprea</imatge>
  <autor>kevin2004</autor>
</articlein>
```
- Response:** 201 Created, 66 ms, 227 B.
- Response Body (JSON):**

```
{
  "id": 4,
  "text": "Aquest es el text de aquest article",
  "titol": "El titol aquest",
  "privacitat": 0,
  "imatge": "julianoprea",
  "autor": "kevin2004"
}
```

Correcte : SI

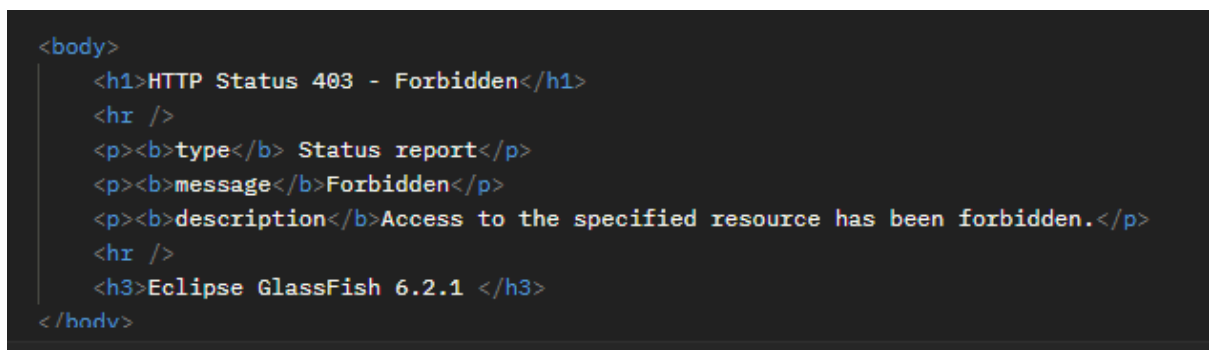
Cas 11

DELETE /rest/api/v1/article/\${id}

Volem fer delete de un article que no es nostre, llavors ens hauria de denegar

Sortida Esperada: Codi 403 fent referencia a “FORBIDDEN” perquè s’ha desitjat borra un article que no es nostre

Sortida Real:



The screenshot shows the raw HTML response for a 403 Forbidden status:

```
<body>
  <h1>HTTP Status 403 - Forbidden</h1>
  <hr />
  <p><b>type</b> Status report</p>
  <p><b>message</b>Forbidden</p>
  <p><b>description</b>Access to the specified resource has been forbidden.</p>
  <hr />
  <h3>Eclipse GlassFish 6.2.1 </h3>
</body>
```

Correcte : SI

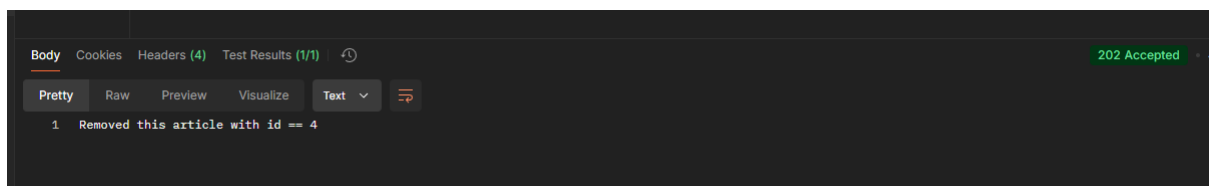
Cas 12

DELETE /rest/api/v1/article/\${id}

Volem fer delete de un article que es nostre llavors hauria de poder fer-lo a la perfecció

Sortida Esperada: Codi 202 fent referencia a “ACCEPTED” perquè s’ha desitjat borra un article que és nostre llavors ho tenim permès.

Sortida Real:



Correcte : SI

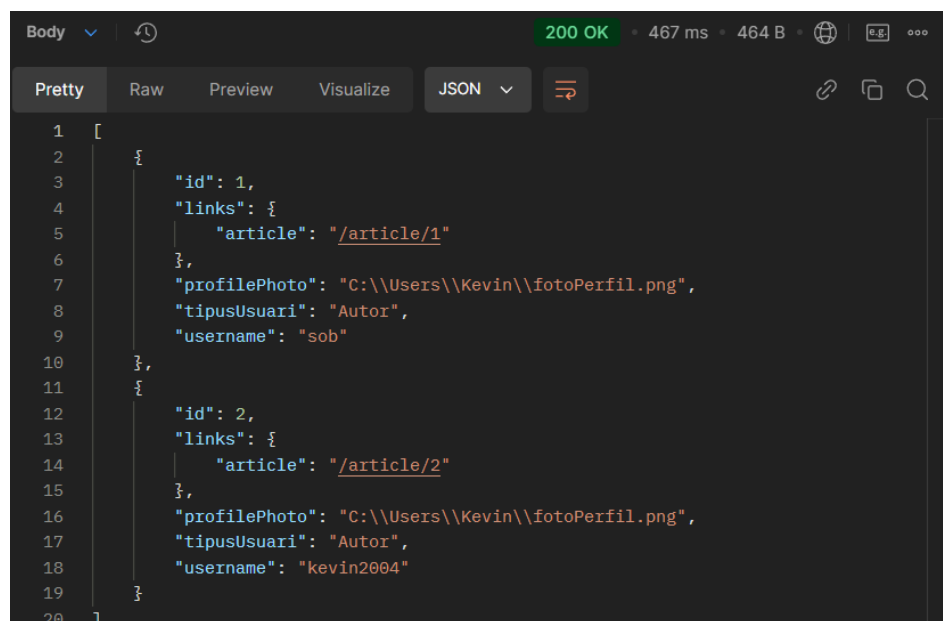
Cas 13

GET /rest/api/v1/customer

Volem obtenir una llista de customers amb informació reduïda en format JSON, ha de mostrar el tipus d'usuari (Lector o Autor) i en el cas que sigui autor el link a l'últim article.

Sortida Esperada: Codi 200 fent referència a "OK", indicant que tot ha anat correcte i ens retorna la informació d'aquests en JSON.

Sortida Real:



```
Body 200 OK • 467 ms • 464 B
Pretty Raw Preview Visualize JSON
1 [
2   {
3     "id": 1,
4     "links": {
5       "article": "/article/1"
6     },
7     "profilePhoto": "C:\\Users\\Kevin\\fotoPerfil.png",
8     "tipusUsuari": "Autor",
9     "username": "sob"
10  },
11  {
12    "id": 2,
13    "links": {
14      "article": "/article/2"
15    },
16    "profilePhoto": "C:\\Users\\Kevin\\fotoPerfil.png",
17    "tipusUsuari": "Autor",
18    "username": "kevin2004"
19  }
20 ]
```

Correcte : SI

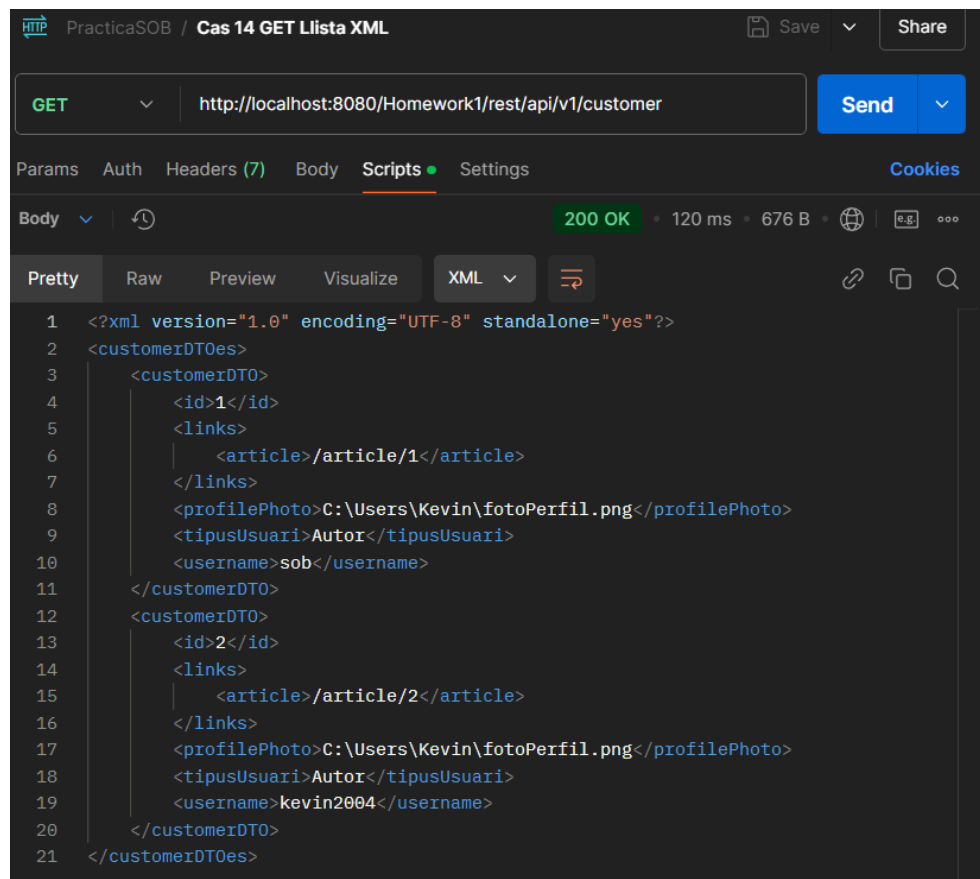
Cas 14

GET /rest/api/v1/customer

Volem obtenir una llista de customers amb informació reduïda en format XML, ha de mostrar el tipus d'usuari (Lector o Autor) i en el cas que sigui autor el link a l'últim article.

Sortida Esperada: Codi 200 fent referència a "OK", indicant que tot ha anat correcte i ens retorna la informació d'aquests en XML. Ens retornarà un objecte de classe CustomerDTO, ja que aquest és el DTO implementat per a la sortida reduïda de informació.

Sortida Real:



```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <customerDTOes>
3   <customerDTO>
4     <id>1</id>
5     <links>
6       <article>/article/1</article>
7     </links>
8     <profilePhoto>C:\Users\Kevin\fotoPerfil.png</profilePhoto>
9     <tipusUsuari>Autor</tipusUsuari>
10    <username>sob</username>
11  </customerDTO>
12  <customerDTO>
13    <id>2</id>
14    <links>
15      <article>/article/2</article>
16    </links>
17    <profilePhoto>C:\Users\Kevin\fotoPerfil.png</profilePhoto>
18    <tipusUsuari>Autor</tipusUsuari>
19    <username>kevin2004</username>
20  </customerDTO>
21 </customerDTOes>
```

Correcte : SI

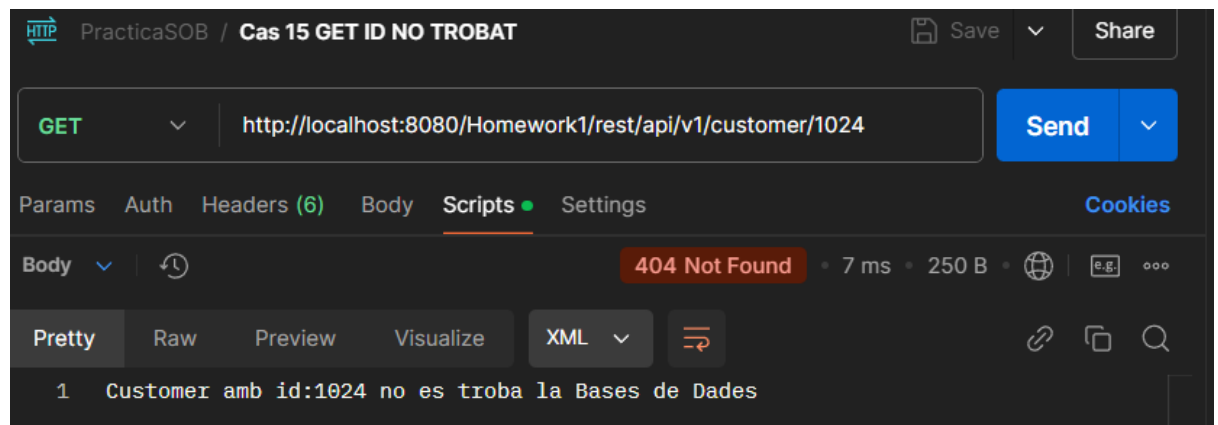
Cas 15

GET /rest/api/v1/customer/1024

En aquest cas estem demanant el customer amb identificador 1024, aquest no existeix a la base de dades, per lo que comprovarem que en el cas que no existeixi el recurs sol·licitat pel client, es retorni una resposta correcta.

Sortida Esperada: Codi 404 fent referencia a “NOT FOUND”, indicant que el recurs no es troba a la base de dades, a més retornarà un missatge especificant això mateix.

Sortida Real:



Correcte : SI

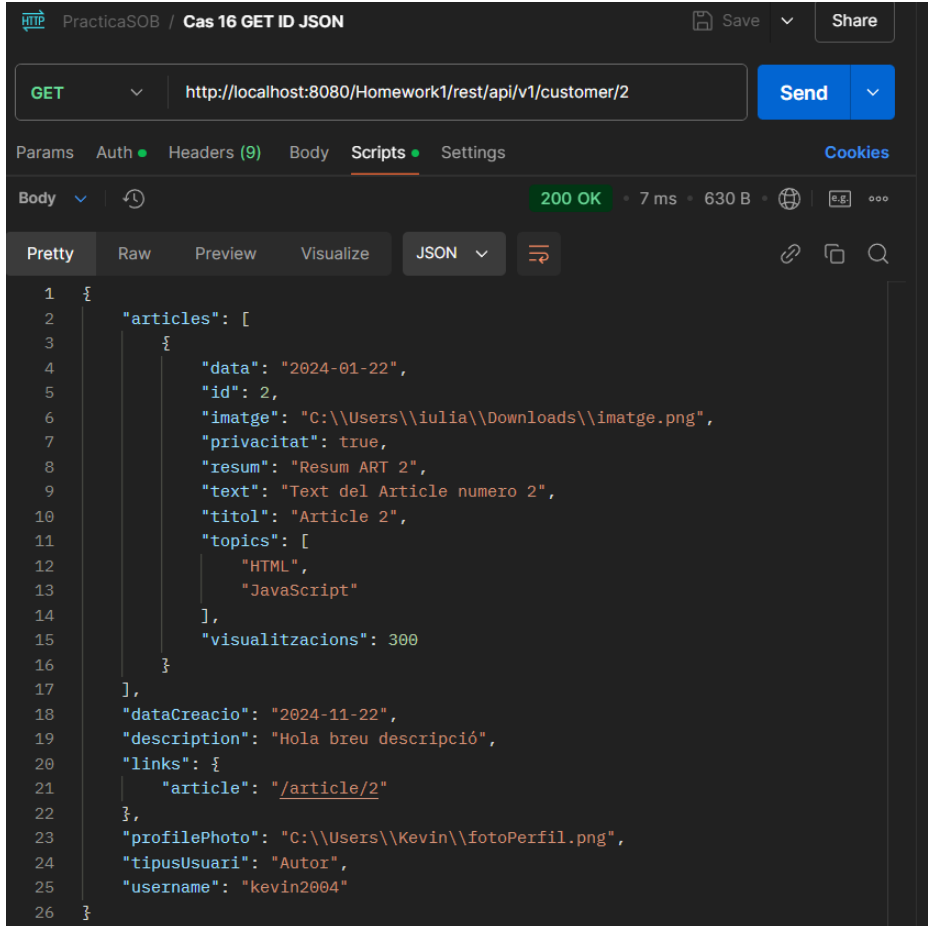
Cas 16

GET /rest/api/v1/customer/2

Volem obtindre la informació específica i detallada del customer amb identificador 2, aquesta informació ha d'estar en format JSON, aquesta petició ens ha de retornar els articles (així com la seva informació), la data de creació, la descripció del perfil, l'últim link al seu últim article, la seva foto de perfil, quin tipus d'usuari és i finalment lo seu username.

Sortida Esperada: Codi 200 fent referencia a “OK”, indicant que el recurs s’ha trobat i per tant mostrarà la seva informació en format JSON.

Sortida Real:



The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:8080/Homework1/rest/api/v1/customer/2
- Status:** 200 OK
- Response Time:** 7 ms
- Response Size:** 630 B
- Response Format:** JSON
- Response Body (Pretty):**

```
1 {
2   "articles": [
3     {
4       "data": "2024-01-22",
5       "id": 2,
6       "imatge": "C:\\Users\\iulia\\Downloads\\imatge.png",
7       "privacitat": true,
8       "resum": "Resum ART 2",
9       "text": "Text del Article numero 2",
10      "titol": "Article 2",
11      "topics": [
12        "HTML",
13        "JavaScript"
14      ],
15      "visualitzacions": 300
16    }
17  ],
18  "dataCreacio": "2024-11-22",
19  "description": "Hola breu descripció",
20  "links": {
21    "article": "/article/2"
22  },
23  "profilePhoto": "C:\\Users\\Kevin\\fotoPerfil.png",
24  "tipusUsuari": "Autor",
25  "username": "kevin2004"
26 }
```

Correcte : SI

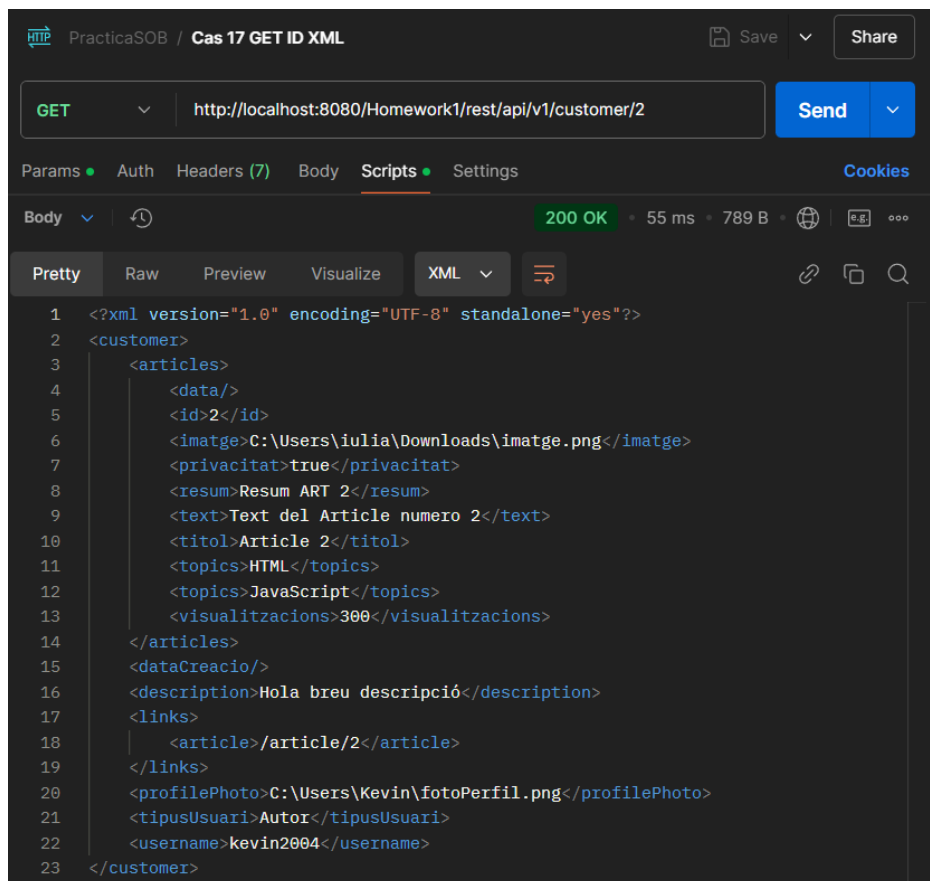
Cas 17

GET /rest/api/v1/customer/2

Volem obtenir la informació específica i detallada del customer amb identificador 2, aquesta informació ha d'estar en format XML, aquesta petició ens ha de retornar els articles (així com la seva informació), la data de creació, la descripció del perfil, l'últim link al seu últim article, la seva foto de perfil, quin tipus d'usuari és i finalment lo seu username.

Sortida Esperada: Codi 200 fent referencia a "OK", indicant que el recurs s'ha trobat i per tant mostrarà la seva informació en format XML.

Sortida Real:



```
HTTP PracticaSOB / Cas 17 GET ID XML
GET http://localhost:8080/Homework1/rest/api/v1/customer/2
Send

Params Auth Headers (7) Body Scripts Settings Cookies
Body 200 OK 55 ms 789 B
Pretty Raw Preview Visualize XML
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <customer>
3   <articles>
4     <data/>
5     <id>2</id>
6     <imatge>C:\Users\iulia\Downloads\imatge.png</imatge>
7     <privacitat>true</privacitat>
8     <resum>Resum ART 2</resum>
9     <text>Text del Article numero 2</text>
10    <titol>Article 2</titol>
11    <topics>HTML</topics>
12    <topics>JavaScript</topics>
13    <visualitzacions>300</visualitzacions>
14  </articles>
15  <dataCreacio/>
16  <description>Hola breu descripció</description>
17  <links>
18    <article>/article/2</article>
19  </links>
20  <profilePhoto>C:\Users\Kevin\fotoPerfil.png</profilePhoto>
21  <tipusUsuari>Autor</tipusUsuari>
22  <username>kevin2004</username>
23 </customer>
```

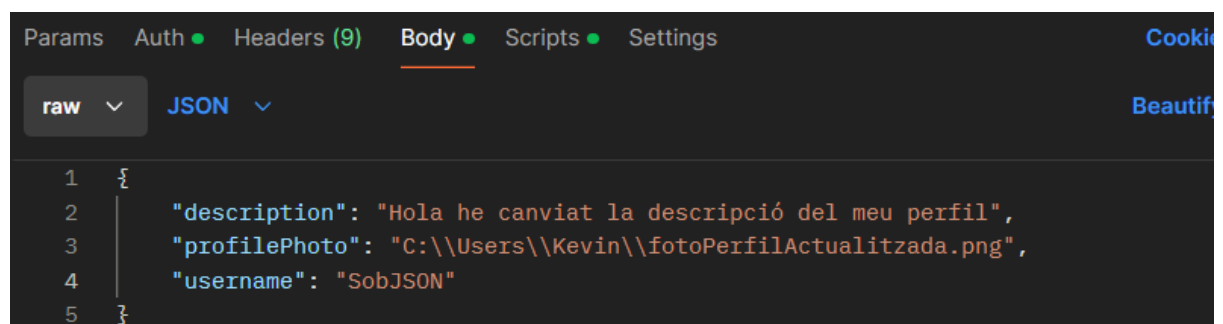
Correcte : SI

Cas 18

PUT /rest/api/v1/customer/1

Ens permet modificar el customer per la seva id, es necessari ficar les credencials per poder fer aquesta actualització de dades, per tant si l'usuari no coincideix en lo customer triat (està registrat però intenta modificar altres dades) sortirà un error. D'altra banda si es correcte actualitzarà les dades i les persistirà.

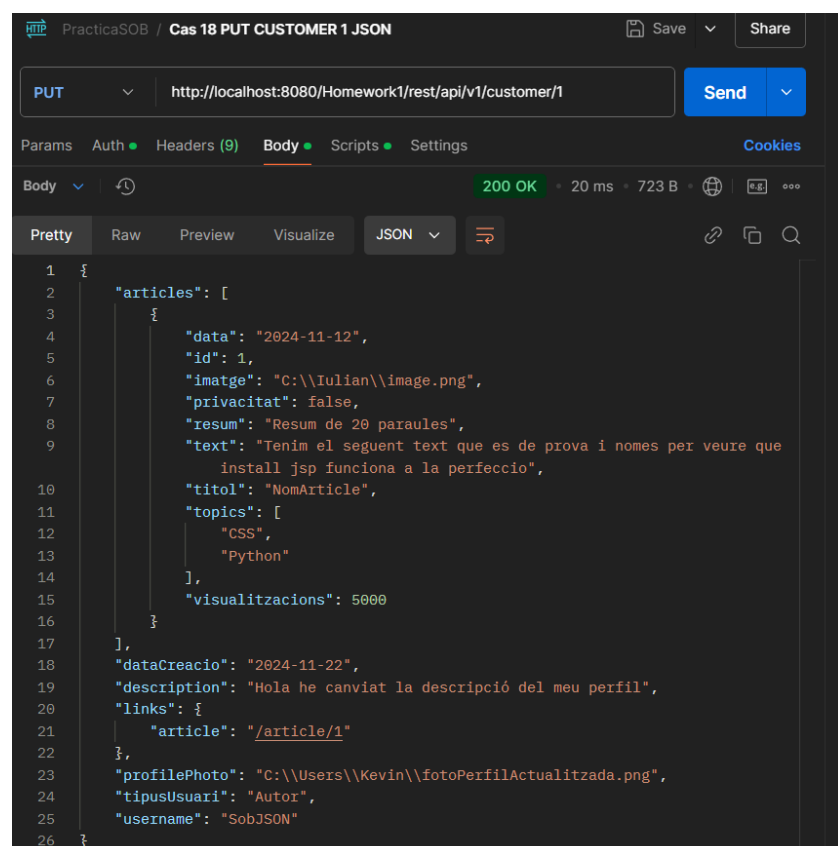
Dades enviades per modificar lo customer.



```
1 {
2   "description": "Hola he canviat la descripció del meu perfil",
3   "profilePhoto": "C:\\Users\\Kevin\\fotoPerfilActualitzada.png",
4   "username": "SobJSON"
5 }
```

Sortida Esperada: Codi 200 fent referencia a “OK”, indicant que el recurs s’ha modificat correctament, a més retornarà l’usuari en JSON en los camps modificats.

Sortida Real:



```
1 {
2   "articles": [
3     {
4       "data": "2024-11-12",
5       "id": 1,
6       "imatge": "C:\\Iulian\\image.png",
7       "privacitat": false,
8       "resum": "Resum de 20 paraules",
9       "text": "Tenim el següent text que es de prova i nomes per veure que
10        install jsp funciona a la perfeccio",
11       "titol": "NomArticle",
12       "topics": [
13         "CSS",
14         "Python"
15       ],
16       "visualitzacions": 5000
17     }
18   ],
19   "dataCreacio": "2024-11-22",
20   "description": "Hola he canviat la descripció del meu perfil",
21   "links": {
22     "article": "/article/1"
23   },
24   "profilePhoto": "C:\\Users\\Kevin\\fotoPerfilActualitzada.png",
25   "tipusUsuari": "Autor",
26   "username": "SobJSON"
27 }
```

Correcte : SI

Cas 19

PUT /rest/api/v1/customer/2

Ens permet modificar el customer per la seva id, es necessari ficar les credencials per poder fer aquesta actualització de dades, per tant si l'usuari no coincideix en lo customer triat (està registrat però intenta modificar altres dades) sortirà un error. D'altra banda si es correcte actualitzarà les dades i les persistirà.

Dades enviades per modificar lo customer.

The screenshot shows a REST client interface with the following components:

- Params**: Tab selected.
- Auth**: Tab selected, showing **Basic Auth** type.
- Username**: Input field with value `kevin2004`.
- Password**: Input field with value `kevin` and a warning icon.
- Body**: Tab selected, showing an XML payload.

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <customer>
3   <description>He aconseguit canviar la descripcio en format XML !!!</
   description>
4   <profilePhoto>C:\Users\Kevin\fotoPerfilActualitzadaXML.png</profilePhoto>
5   <username>SobXML</username>
6 </customer>
```

Sortida Esperada: Codi 200 fent referencia a "OK", indicant que el recurs s'ha modificat correctament, a més retornarà l'usuari en XML en los camps modificats.

Sortida Real:

PracticaSOB / Cas 19 PUT CUSTOMER 2 XML

PUT http://localhost:8080/Homework1/rest/api/v1/customer/2 Send

Params Auth Headers (10) Body Scripts Settings Cookies

Body 200 OK • 38 ms • 833 B

Pretty Raw Preview Visualize XML

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <customer>
3   <articles>
4     <data/>
5     <id>2</id>
6     <imatge>C:\Users\iulia\Downloads\imatge.png</imatge>
7     <privacitat>true</privacitat>
8     <resum>Resum ART 2</resum>
9     <text>Text del Article numero 2</text>
10    <titol>Article 2</titol>
11    <topics>HTML</topics>
12    <topics>JavaScript</topics>
13    <visualitzacions>300</visualitzacions>
14  </articles>
15  <dataCreacio/>
16  <description>He aconseguit canviar la descripcio en format XML !!!</description>
17  <links>
18    <article>/article/2</article>
19  </links>
20  <profilePhoto>C:\Users\Kevin\fotoPerfilActualitzadaXML.png</profilePhoto>
21  <tipusUsuari>Autor</tipusUsuari>
22  <username>SobXML</username>
23 </customer>
```

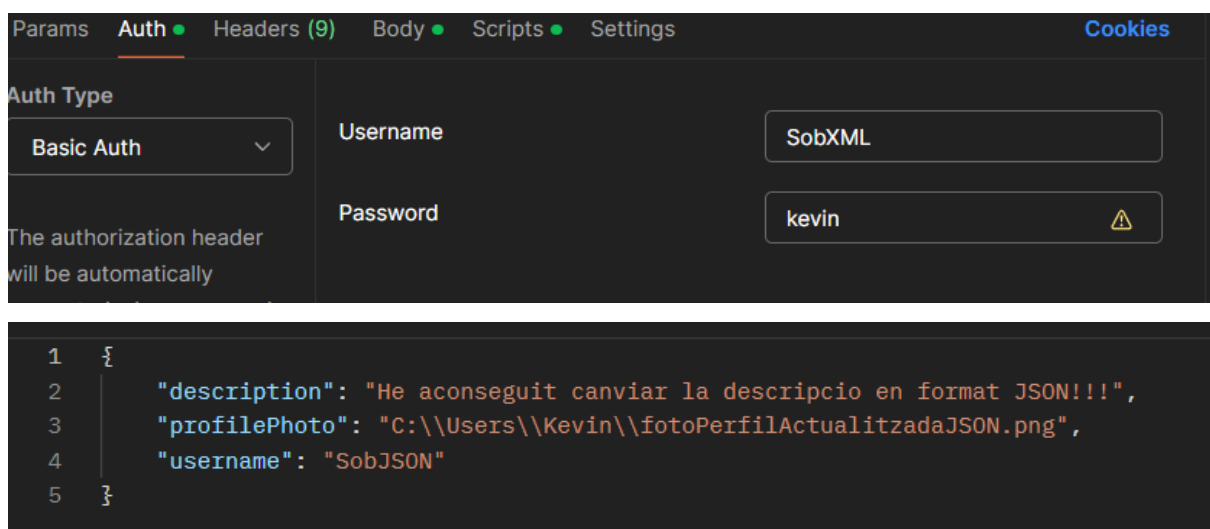
Correcte : SI

Cas 20

PUT /rest/api/v1/customer/1

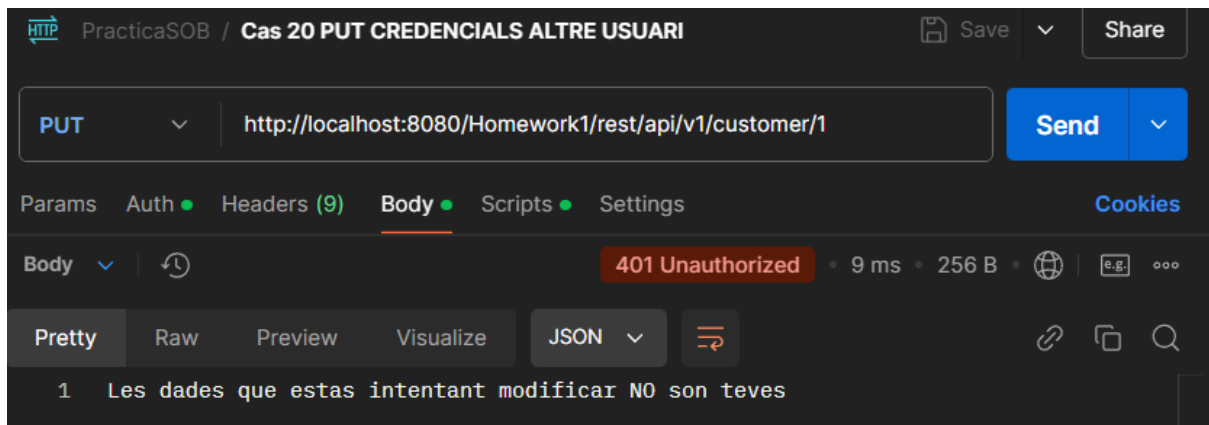
Ens permet modificar el customer per la seva id, es necessari ficar les credencials per poder fer aquesta actualització de dades, per tant si l'usuari no coincideix en lo customer triat (està registrat però intenta modificar altres dades) sortirà un error. D'altra banda si es correcte actualitzarà les dades i les persistirà.

Dades enviades per modificar lo customer.



Sortida Esperada: Codi 401 fent referencia a “UNAUTHORIZED”, indicant que el recurs no s’ha modificat ja que no ens pertanyen aquestes dades. A més mostrarà un missatge explicant el motiu de l’error.

Sortida Real:



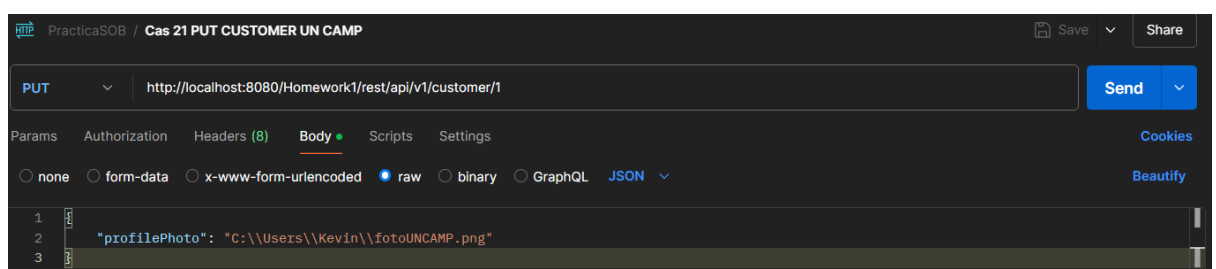
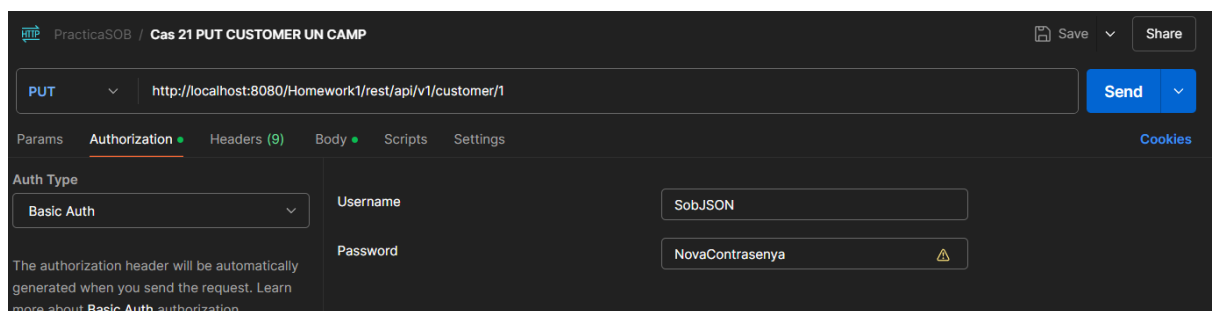
Correcte: SI

Cas 21

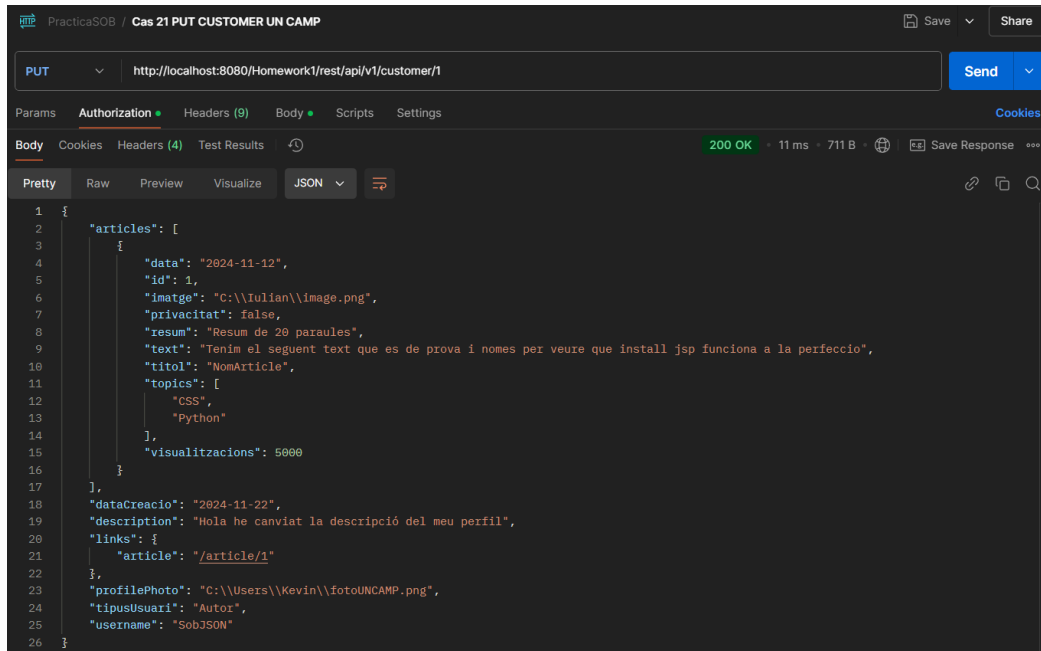
PUT /rest/api/v1/customer/1

Modificarem només un camp del customer, mantenint les dades anteriors sense modificar-les, d'aquesta forma no cal haver de introduir novament totes les dades a l'hora de fer la crida a aquesta petició. A més, es farà servir les credencials que han sigut actualitzades al cas 18.

Dades enviades per modificar lo customer.



Sortida Esperada: Codi 200 fent referencia a “OK”, indicant que el recurs s’ha actualitzat, ha de mostrar la informació del customer sense modificacions, excepte l’actualització de la foto de perfil.



The screenshot shows a REST client interface with the following details:

- Method:** PUT
- URL:** http://localhost:8080/Homework1/rest/api/v1/customer/1
- Status:** 200 OK
- Response Body (JSON):**

```
1 {
2   "articles": [
3     {
4       "data": "2024-11-12",
5       "id": 1,
6       "imatge": "C:\\Iulian\\image.png",
7       "privacitat": false,
8       "resum": "Resum de 20 paraules",
9       "text": "Tenim el següent text que es de prova i nomes per veure que install.jsp funciona a la perfeccio",
10      "titol": "NomArticle",
11      "topics": [
12        "CSS",
13        "Python"
14      ],
15      "visualitzacions": 5000
16    }
17  ],
18  "dataCreacio": "2024-11-22",
19  "description": "Hola he canviat la descripció del meu perfil",
20  "links": {
21    "article": "/article/1"
22  },
23  "profilePhoto": "C:\\Users\\Kevin\\fotoUNCAMP.png",
24  "tipusUsuari": "Autor",
25  "username": "SobJSON"
26 }
```

Correcte: SI

5. Conclusions

Després d'haver realitzat la pràctica hem arribat a diferents conclusions.

Primerament, és essencial tindre una API RESTful ben estructurada per al desenvolupament del sistema, a més això ens permet fer un manteniment i una extensió d'aquests sense molt de problema.

Com segon punt, ens hem adonat que és necessari implementar mecanisme d'autenticació adequat per la protecció dels nostres recursos. A més, cal recordar l'eficiència de les nostres operacions amb JPA, evitant consultes que poden ser costes si no són necessàries.

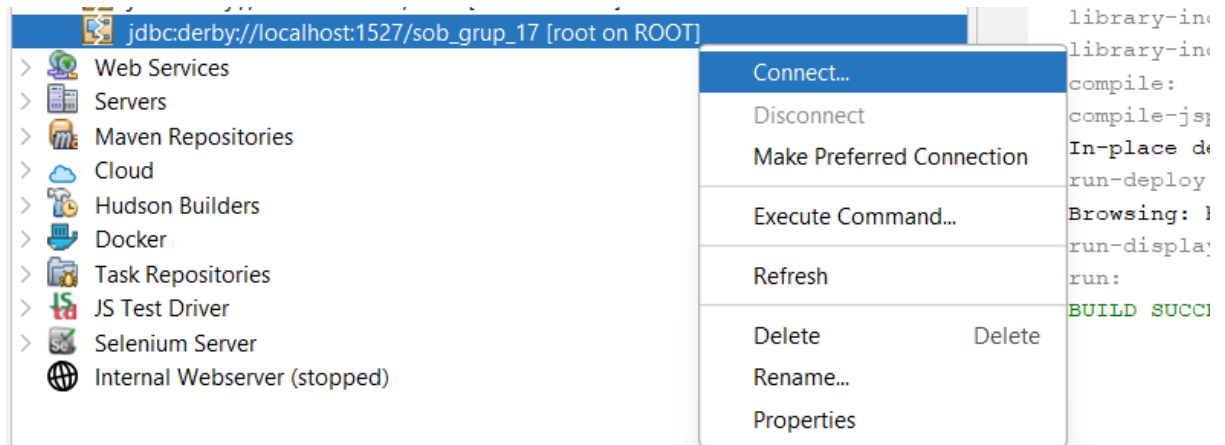
Un altre aspecte fonamental són les proves al nostre sistema. S'ha de dissenyar un joc de proves que abastin tots els casos tant positius com negatius, aplicant metodologies TDD (Test-Driven Development) o BDD (Behavior-Driven Development).

També, hem pogut comprendre la importància de l'estàndard HTTP, ja que aprofitem els codis d'estat com poden ser 200 (OK) o 404 (Not Found) i aplicar els principis HATEOAS, per millorar la comunicació entre client-servidor. Amb tot això, es complementen la flexibilitat del format de les dades, fent servir XML i JSON augmentat així la interoperabilitat amb diferents sistemes.

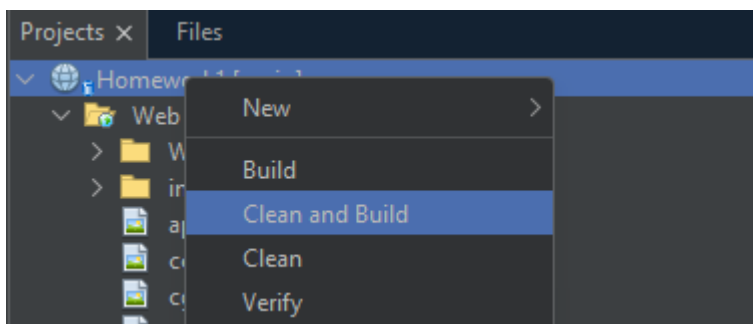
Finalment, hem entès l'important que és tindre la definició dels recursos clarament separats, en el nostre cas articles i customers. Ja que d'aquesta forma podem fer una bona separació de responsabilitats i escalabilitat d'ambdós.

6. Manual d'instal·lació

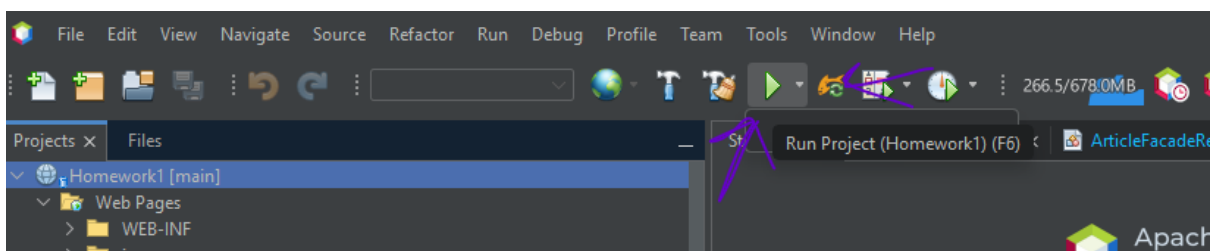
1. Obrim **Apache Netbeans 18**
2. Obrim el projecte **Homework1**
3. Ens connectem a la base de dades **sob_grup_17** (usuari i contrasenya: root)



4. Fem un “**clean and build**”



5. Fem un “**deploy and run**”

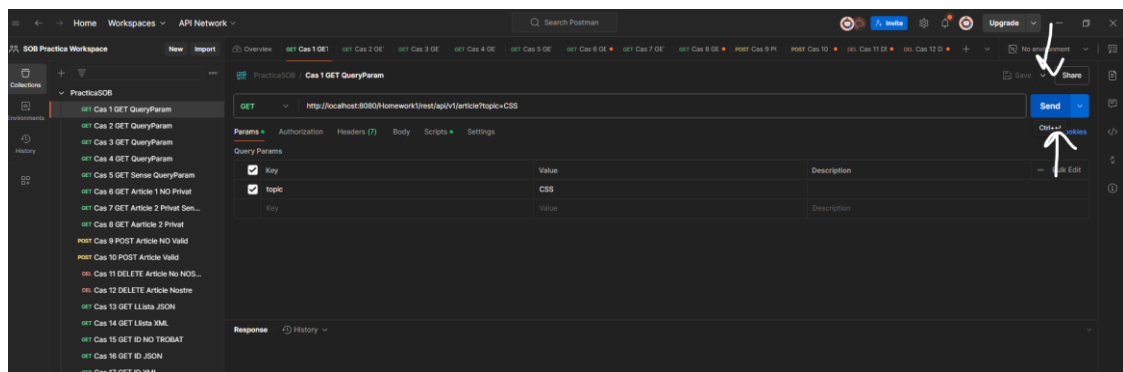


6. Donem “**click**” al boto de **insert**

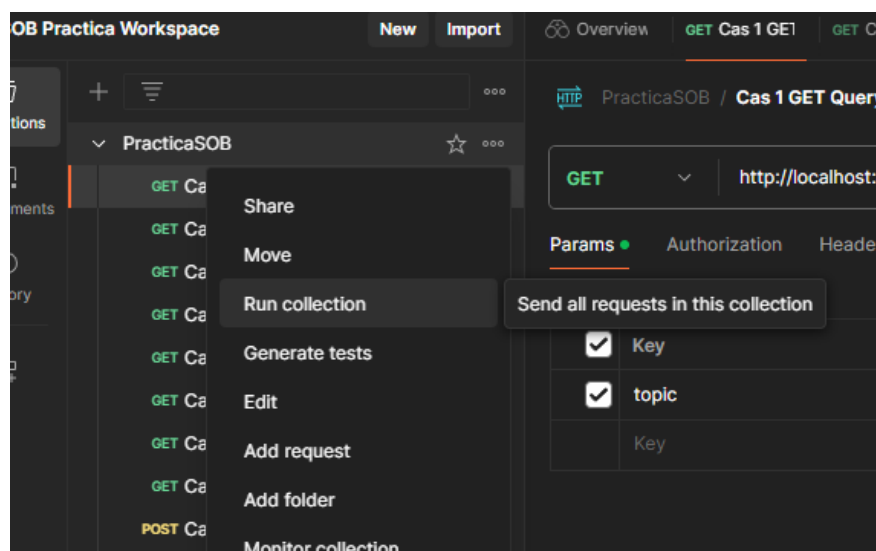


7. Seguit aquest pas obrim **l'aplicació de Postman** amb el testos especificats, amb el enllaç següent: <https://sob-practica.postman.co/workspace/SOB-Practica-Workspace~2cbb3e2a-cb7c-411a-ace4-9dd2db930f49/collection/39916940-63a4f6e0-e3c5-463d-84b3-7d44fb13ba89?action=share&creator=39916940>
8. Anem fent **send** al diferents mètodes seguint el mateix que els casos de joc de proves especificat o podem executar tots el testos amb un delay de 1000 ms per a que no hi hagi interferències.

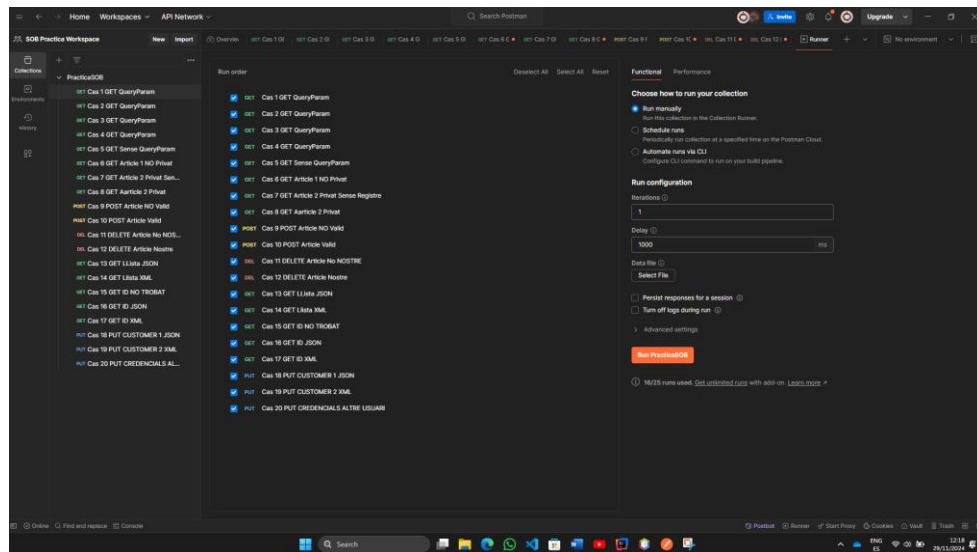
SEND:



Test Run de la Collection:



Fiquem el retard especificat en la practica en el manual de instal·lació i podem donar run PracticaSOB



7. Video

Enllaç al vídeo explicatiu sobre la pràctica: [Julian Kevin SOB Prac1.mp4](#)