

A5 - Web Services

Daniel Sánchez
Edgar Moreno

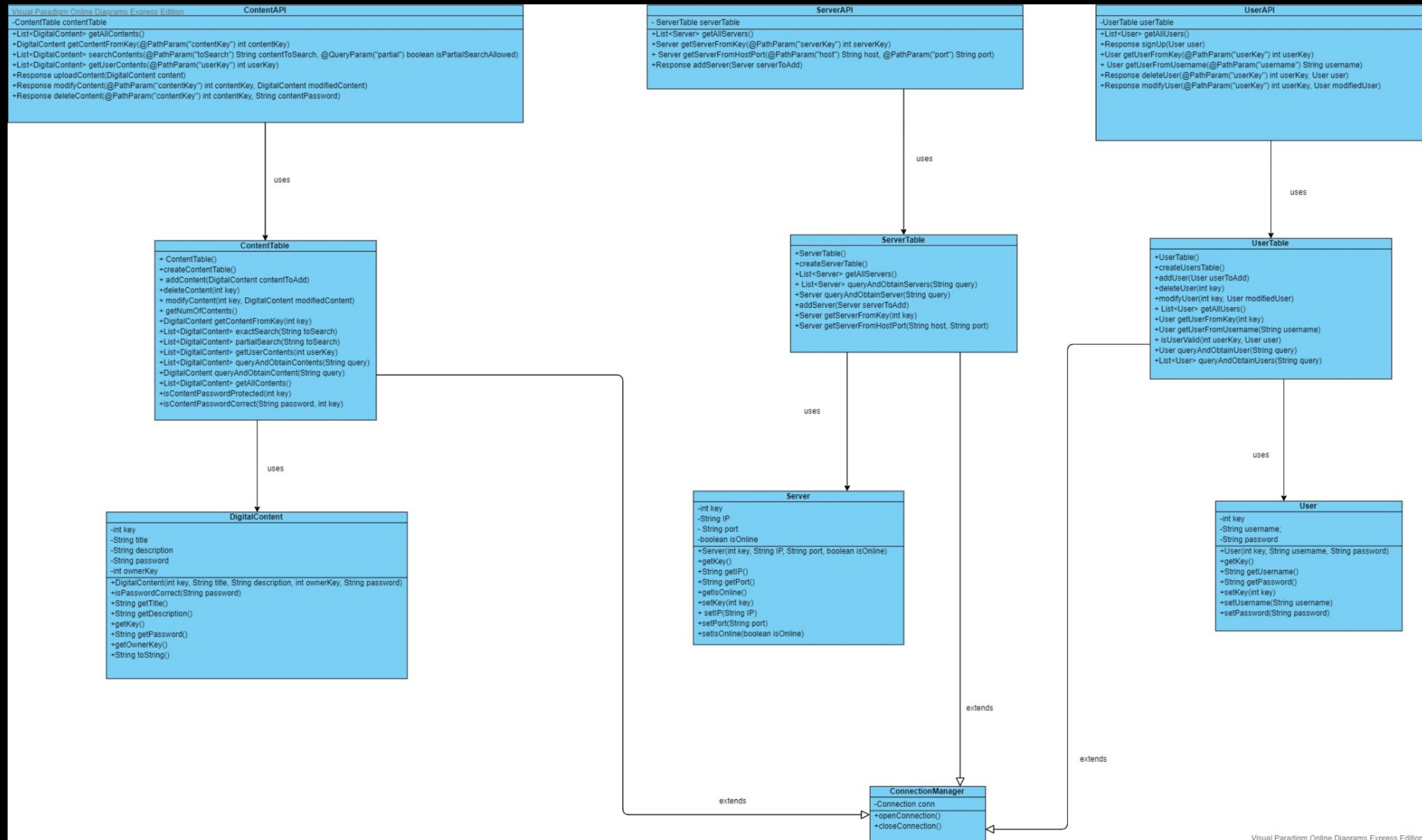
entities

contents

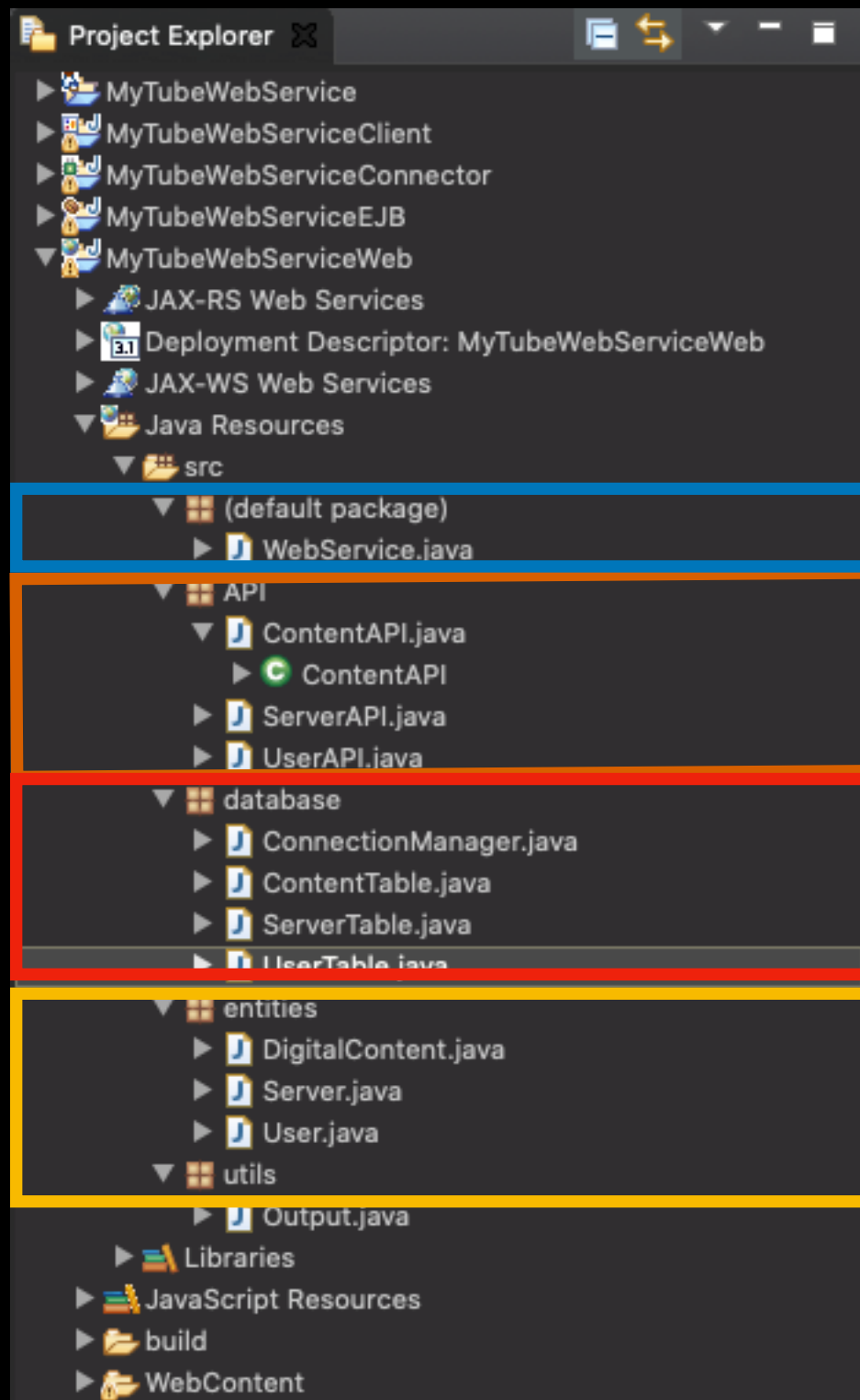
users

servers

uml



project structure



WebService.java

defines the `@ApplicationPath (/api)`

API related classes

define the paths and operations for the different entities, triggered by Java clients or direct access from browser

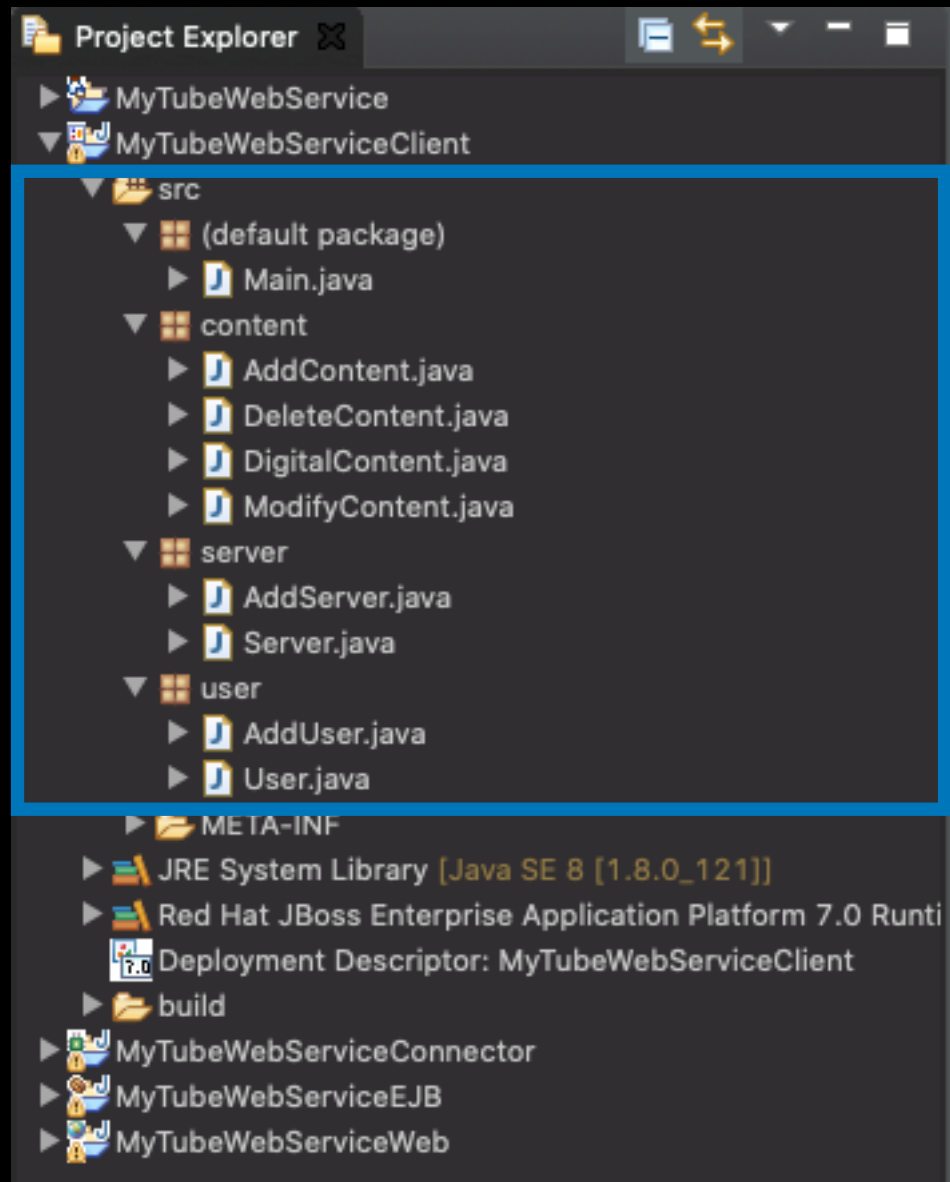
Database related classes

define the operations for the different databases, triggered by the API related classes

Auxiliar classes

define operations for packing, unpacking and formatting purposes

project structure



Different clients for PUT, POST and DELETE operations

GET operations can be easily triggered directly from the browser or using tools like curl

endpoints > contents

endpoint	description	returning value
GET /content	gets all contents	“application/json”
GET /content/{contentKey}	gets content from key	“application/json”
GET /content/user/{userKey}	gets user’s contents	“application/json”
GET /content/server/{serverKey}	gets server’s contents	“application/json”
GET /content/search/{toSearch}?partial=False	exact/partial search on content’s description and password. partial is optional and False by default	“application/json”

endpoints > contents

endpoint	description	returning value
POST /content	creates content	201 (if added in database), 409 (if content's title already exists) or 500 (if can't be added)
PUT /content/{contentKey}	modifies content (must provide the correct content password)	201 (if modified), 401 (if content's password is incorrect), 409 (if content does not exist or 500 (if can't be modified)
DELETE /content/{contentKey}	deletes content (must provide the correct content password)	200 (if deleted), 401 (if content's password is incorrect), 409 (if content does not exist or 500 (if can't be deleted)

endpoints > users

endpoint	description	returning value
GET /user	gets all users	"application/json"
GET /user/{userKey}	gets user from key	"application/json"
GET /user/username/{username}	gets user from username	"application/json"
POST /user	creates user	201 (if added in database), 409 (if username is taken) or 500 (if can't be added)

endpoints > servers

endpoint	description	returning value
GET /server	gets all servers	“application/json”
GET /server/{serverKey}	gets server from key	“application/json”
GET /server/host/{host}/port/{port}	gets server from host and port	“application/json”
POST /server	creates server	201 (if added in database), 409 (if server with same host:port already exists) or 500 (if can't be added)

main use cases

- In order to start adding contents we need to add servers and users first. The contents cannot exist without users that own them and servers which contain them
- To **add a server** we can execute **AddServer.java** contained in *MyTubeWebServiceClient*, which will POST to *http://localhost:8080/MyTubeWebServiceWeb/api/server*
- The execution will add a server in the table named servers located in the database named Postgres. The server will have IP “127.0.0.1” and port “1234” (view **AddServer.java**)
- The return code will be 201. We can see the execution output in the next slide;

main use cases

```
AddServer.java
12 public static void main(String[] args) {
13     //POST
14     try {
15         URL url = new URL ("http://localhost:8080/MyTubeWebServiceWeb/api/server");
16         HttpURLConnection conn = (HttpURLConnection) url.openConnection();
17         conn.setDoOutput(true);
18         conn.setRequestMethod("POST");
19         conn.setRequestProperty("Content-Type", "application/json");
20
21         // server to add - pass it as bytes from JSON
22         Server toAdd = new Server("127.0.0.127", "1234");
23         System.out.println("Adding server:" + toAdd.toJson());
24         OutputStream os = conn.getOutputStream();
25         os.write(toAdd.toJson().getBytes());
26         os.flush();
27
28         if(conn.getResponseCode() != 201) {
29             throw new RuntimeException("Failed: HTTP error code: " + conn.getResponseCode());
30         }
31         System.out.println("Success: HTTP code: " + conn.getResponseCode());
32
33         conn.disconnect();
34
35     } catch (MalformedURLException e) {
36     }
37 }
```

Console

<terminated> AddServer [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/bin/java (Dec 19, 2019, 6:03:13 PM)

Adding server:{"key":0,"port":"1234","isOnline":false, "ip":"127.0.0.127"}

Success: HTTP code: 201

to create a server
POST /server

```
http://localhost:8080/MyTubeWebServiceWeb/api/server
http://localhost:8080/MyTubeWebServiceWeb/api/server

[{"key":1,"port":"1234","isOnline":false,"ip":"127.0.0.127"}]
```

to get all servers info
GET /server

main use cases

- To **add a user** we can execute ***AddUser.java*** contained in *MyTubeWebServiceClient*, which will POST to <http://localhost:8080/MyTubeWebServiceWeb/api/user>
- The execution will add a user in the table named users located in the database named Postgres. The user will have username “genericusername” and password “genericpassword” (view ***AddUser.java***)
- *The return code will be 201. We can see the execution output in the next slide;*

main use cases

The image displays two side-by-side screenshots illustrating the main use cases of a web service.

Left Screenshot (Java IDE): Shows the source code for `AddUser.java`. The `main` method performs a POST request to `http://localhost:8080/MyTubeWebServiceWeb/api/user` with a JSON body: `{ "key": 0, "username": "genericuser", "password": "genericpassword" }`. The console output shows: `Adding user: {"key":0,"username":"genericuser","password":"genericpassword"}` and `Success: HTTP code: 201`.

Right Screenshot (Web Browser): Shows the response from the GET endpoint at `http://localhost:8080/MyTubeWebServiceWeb/api/user`. The response is a JSON array: `[{"key":1,"username":"genericuser","password":"genericpassword"}]`.

to create a user
POST /user

to get all users info
GET /user

main use cases

- We can **add a content** by executing the client *AddContent.java* contained in *MyTubeWebServiceClient*
- The execution will add a content with title “title12” and description “description” with userOwnerKey 1 and serverOwnerKey 1
- *The return code will be 201. We can see the execution output in the next slide;*

main use cases

```
AddContent.java
14 public static void main(String[] args) {
15     //POST
16     try {
17         URL url = new URL ("http://localhost:8080/MyTubeWebServiceWeb/api/co
18         HttpURLConnection conn = (HttpURLConnection) url.openConnection();
19         conn.setDoOutput(true);
20         conn.setRequestMethod("POST");
21         conn.setRequestProperty("Content-Type", "application/json");
22
23         // content to add - pass it as bytes from JSON
24         DigitalContent toAdd = new DigitalContent("title12345", "description
25         System.out.println("Adding content:" + toAdd.toJson());
26         OutputStream os = conn.getOutputStream();
27         os.write(toAdd.toJson().getBytes());
28         os.flush();
29
30         if(conn.getResponseCode() != HttpURLConnection.HTTP_CREATED) {
31             throw new RuntimeException("Failed: HTTP error code: " + conn.ge
32         }
33
34         System.out.println("Success: HTTP code: " + conn.getResponseCode());
35         conn.disconnect();
36
37     } catch (MalformedURLException e) {
```

```
Console
<terminated> AddContent (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/bin/java (Dec 19, 2019, 6:26:20 PM)
Adding content:{"key":0,"title":"title12345","description":"description","password":"","userOwnerKey":1,"serverOwnerKey":1}
Success: HTTP code: 201
```

```
http://localhost:8080/MyTubeWebServiceWeb/api/content
http://localhost:8080/MyTubeWebServiceWeb/api/co

{"key":4,"title":"title12345","description":"description","password":"","userOwnerKey":1,"serverOwnerKey":1}
```

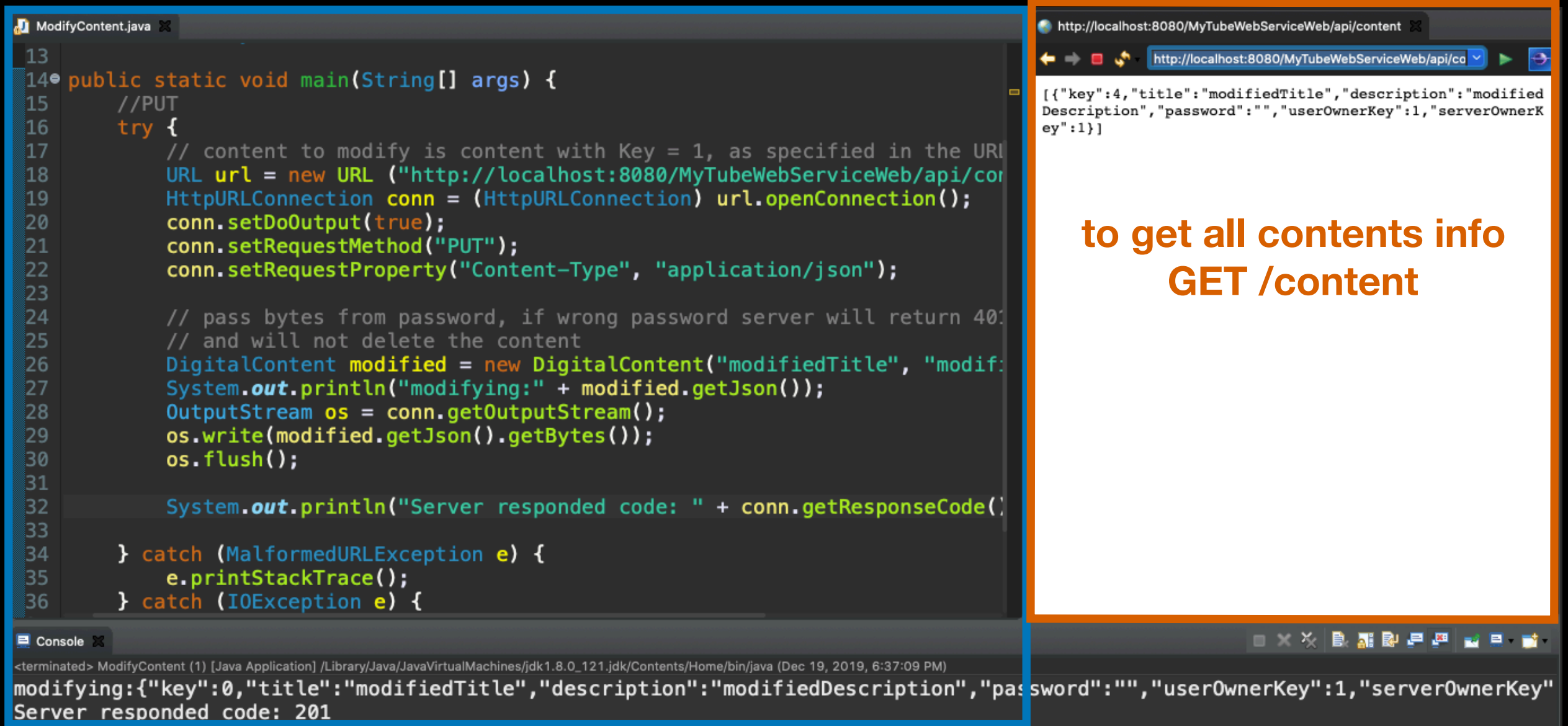
to get all contents info
GET /content

to create a content
POST /content

main use cases

- We can **modify a content** by executing the client ***ModifyContent.java*** contained in *MyTubeWebServiceClient*
- The execution will modify content with key 4 and password “”. The password we pass does not matter as there is no password for the content
- *The return code will be 201. We can see the execution output in the next slide;*

main use case



The image shows a Java IDE on the left and a web browser on the right. The IDE window, titled 'ModifyContent.java', displays a Java program that sends a PUT request to a web service. The code includes comments and uses the `URLConnection` class to send the request. The console at the bottom shows the output of the program, including the JSON body of the request and the server's response code (201). The web browser window, titled 'http://localhost:8080/MyTubeWebServiceWeb/api/content', shows the JSON response from the server, which includes details about the modified content.

```
13
14 public static void main(String[] args) {
15     //PUT
16     try {
17         // content to modify is content with Key = 1, as specified in the URL
18         URL url = new URL ("http://localhost:8080/MyTubeWebServiceWeb/api/content");
19         HttpURLConnection conn = (HttpURLConnection) url.openConnection();
20         conn.setDoOutput(true);
21         conn.setRequestMethod("PUT");
22         conn.setRequestProperty("Content-Type", "application/json");
23
24         // pass bytes from password, if wrong password server will return 401
25         // and will not delete the content
26         DigitalContent modified = new DigitalContent("modifiedTitle", "modifiedDescription");
27         System.out.println("modifying:" + modified.getJson());
28         OutputStream os = conn.getOutputStream();
29         os.write(modified.getJson().getBytes());
30         os.flush();
31
32         System.out.println("Server responded code: " + conn.getResponseCode());
33
34     } catch (MalformedURLException e) {
35         e.printStackTrace();
36     } catch (IOException e) {
37         e.printStackTrace();
38     }
39 }
```

modifying:{"key":0,"title":"modifiedTitle","description":"modifiedDescription","password":"","userOwnerKey":1,"serverOwnerKey":1}
Server responded code: 201

http://localhost:8080/MyTubeWebServiceWeb/api/content
[{"key":4,"title":"modifiedTitle","description":"modifiedDescription","password":"","userOwnerKey":1,"serverOwnerKey":1}]

to get all contents info
GET /content

to modify a content
PUT /content

main use cases

- We can **delete a content** by executing the client *DeleteContent.java* contained in *MyTubeWebServiceClient*
- The execution will delete content with key 4 and password “”. The password we pass does not matter as there is no password for the content
- *The return code will be 200. We can see the execution output in the next slide;*

main use case

The image shows a Java IDE on the left and a web browser on the right. The IDE displays a Java class named `DeleteContent` with a `main` method that sends a DELETE request to `http://localhost:8080/MyTubeWebServiceWeb/api/content`. The console shows the response code 200. The browser shows the same URL with an empty response body.

```
9
10 public class DeleteContent {
11
12     public static void main(String[] args) {
13         //DELETE
14         try {
15             // content to delete is content with Key = 4, as specified in the
16             URL url = new URL ("http://localhost:8080/MyTubeWebServiceWeb/api
17             HttpURLConnection conn = (HttpURLConnection) url.openConnection()
18             conn.setDoOutput(true);
19             conn.setRequestMethod("DELETE");
20
21             // pass bytes from password, if wrong password server will return
22             // and will not delete the content
23             OutputStream os = conn.getOutputStream();
24             os.write("1".getBytes());
25             os.flush();
26
27             System.out.println("Server responded: " + conn.getResponseCode())
28             conn.disconnect();
29
30         } catch (MalformedURLException e) {
31             e.printStackTrace();
32         }
33     }
34 }
```

Console: <terminated> DeleteContent (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/bin/java (Dec 19, 2019, 6:38:20 PM)
Server responded: 200

Browser: http://localhost:8080/MyTubeWebServiceWeb/api/content
[]

to get all contents info
GET /content

to delete a content
PUT /content

