

Laborator 4 :

Cerere pentru adaugare de utilizator nou

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer:** Shows the project structure under "SQLProject".
- Code Editor:** Displays a Python file with code for creating a user. The code imports `User` from `models.user_orm` and uses SQLAlchemy's `Session` to add a new user to the database.
- Terminal:** Shows the command-line output of running the script, indicating successful connection to the database and creation of a user named "stefan".
- Postman:** A browser-based API testing tool is open, showing a POST request to "http://127.0.0.1:8000". The request body contains XML for creating a user with the username "stefan" and password "Bond".
- Status Bar:** Shows the current environment as "powershell" and the Python version as "3.9.6 64-bit".

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer:** Shows the project structure under "SQLProject".
- Code Editor:** Displays a Python file with code for creating a user. The code imports `User` from `models.user_orm` and uses SQLAlchemy's `Session` to add a new user to the database.
- Terminal:** Shows the command-line output of running the script, indicating successful connection to the database and creation of a user named "stefan".
- MySQL Workbench:** A separate window shows a table named "roles" with one row: "id" (int(6)) = 1, "name" (varchar(50)) = "admin".
- MySQL Workbench:** Another window shows a table named "users" with several rows, including "id" (int(6)), "username" (varchar(50)), and "password" (varchar(50)).
- Status Bar:** Shows the current environment as "powershell" and the Python version as "3.9.6 64-bit".

Cerere pentru a vedea toti utilizatorii

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure under `SOURCEPROJECT`, including files like `role_repository.py`, `soap_main.py`, and `sql_base.py`.
- Scratch Pad:** A sidebar containing a list of available SOAP operations: `get users`, `create user`, `get user by id`, `get role by id`, `get users info`, `display roles`, `add role`, `change password`, `delete user`, `get role by name`, `login`, `create user role`, and `delete user role`.
- Postman-like Interface:** A central panel for sending POST requests to `http://127.0.0.1:8000`. The current request is `POST display users info`. The body contains a sample XML envelope for the `display_users_info` operation.
- Terminal:** Shows log output from the application, including the start message and two test POST requests.

```

INFO:root:listening to http://127.0.0.1:8000
INFO:root:ws is at: http://127.0.0.1:8000/ws
127.0.0.1 - - [22/Jan/2023 19:22:54] "POST / HTTP/1.1" 200 312
4 - test3 - test3 - []
5 - test - test - []
6 - stefan - parola - []
22 - test1 - test1 - []
23 - test2 - test2 - []
25 - test3 - test3 - []
26 - James - Bond - []
127.0.0.1 - - [22/Jan/2023 19:29:18] "POST / HTTP/1.1" 200 237
127.0.0.1 - - [22/Jan/2023 19:29:18] "POST / HTTP/1.1" 200 237
    
```

Cerere pentru a vedea toate rolurile disponibile

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure under `SOURCEPROJECT`, including files like `role_repository.py`, `soap_main.py`, and `sql_base.py`.
- Scratch Pad:** A sidebar containing a list of available SOAP operations: `get username by id`, `add user`, `get role by id`, `get users info`, `display roles`, `display roles`, `add role`, `change password`, `delete user`, `get role by name`, `login`, `create user role`, and `delete user role`.
- Postman-like Interface:** A central panel for sending POST requests to `http://127.0.0.1:8000`. The current request is `POST display roles`. The body contains a sample XML envelope for the `display_roles_info` operation.
- Terminal:** Shows log output from the application, including the start message and a single test POST request.

```

INFO:root:listening to http://127.0.0.1:8000
INFO:root:ws is at: http://127.0.0.1:8000/ws
127.0.0.1 - - [22/Jan/2023 19:29:18] "POST / HTTP/1.1" 200 237
1 - admin
2 - content manager
3 - artist
4 - client
127.0.0.1 - - [22/Jan/2023 19:31:20] "POST / HTTP/1.1" 200 237
    
```

Cererere pentru a schimba parola unui utilizator

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure with files like `role_repository.py`, `soap_main.py`, and `sql_base.py`.
- Code Editor:** Displays Python code for a `UserRepository` class. The `change_pass` method is highlighted.
- Terminal:** Shows command-line logs for a client application interacting with the service.
- Postman:** An open browser window titled "user_repository.py - SQLProject - Visual Studio Code" displays a POST request to "http://127.0.0.1:8000". The request body is XML for changing a password, and the response shows a successful 200 OK status.

```
1 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:tns="services.user">
2   <soapenv:Body>
3     <ns1:change_pass>
4       <sample:id=26</sample:id>
5       <sample:password>8987</sample:password>
6     </ns1:change_pass>
7   </soapenv:Body>
8 </soapenv:Envelope>
```

Cerere pentru a sterge un utilizator

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure with files like `role_repository.py`, `soap_main.py`, and `sql_base.py`.
- Code Editor:** Displays Python code for a `UserRepository` class. The `delete_user` method is highlighted.
- Terminal:** Shows command-line logs for a client application interacting with the service.
- Postman:** An open browser window titled "user_repository.py - SQLProject - Visual Studio Code" displays a POST request to "http://127.0.0.1:8000". The request body is XML for deleting a user, and the response shows a successful 200 OK status.

```
1 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:tns="services.user">
2   <soapenv:Body>
3     <ns1:delete_user>
4       <sample:id=26</sample:id>
5     </ns1:delete_user>
6   </soapenv:Body>
7 </soapenv:Envelope>
```

Creeare de login a unui utilizator

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure under `SOURCEPROJECT`, including files like `_pycache_`, `base`, `models`, `repositories`, `service`, and `main.py`.
- Code Editor:** Displays Python code for a `UserRepository` class. The `create_user` method takes `username` and `password` parameters and adds a new user to the database. The `login` method takes `username` and `password` and returns a session object.
- Terminal:** Shows command-line logs for testing the application.
- Postman:** An open browser window titled "user_repository.py - SQLProject - Visual Studio Code" displays a POST request to `/login`. The request body contains XML for logging in with username "stefan" and password "parola". The response body shows a successful login response with session information.

Cerere pentru adaugare rol pentru un utilizator

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure under `SOURCEPROJECT`, including files like `_pycache_`, `base`, `models`, `repositories`, `service`, and `main.py`.
- Code Editor:** Displays Python code for a `UserRepository` class. The `create_user` method adds a new user. The `create_user_role` method adds a role to an existing user's session.
- Terminal:** Shows command-line logs for testing the application.
- Postman:** An open browser window titled "user_repository.py - SQLProject - Visual Studio Code" displays a POST request to `/create_user_role`. The request body contains XML for adding a role named "admin" to user "test1". The response body shows a successful role addition response with session information.

Cerere pentru stergere rol pentru un utilizator

The screenshot shows a Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure under "user_repository.py - SQL Project". It includes files like `_pycache_`, `base`, `models`, `repositories`, `service`, and `user_repository.py`. The `user_repository.py` file is open, displaying code for User and Session management.
- Terminal View:** Shows log entries from a server at `127.0.0.1` for various API requests, including POST operations for creating users and deleting user roles.
- Postman View:** Displays a POST request to `http://127.0.0.1:8000` to delete a user role. The request body is an XML envelope containing a `<tns:delete_user_role>` node with a `sampleId` attribute set to `4`.
- Status Bar:** Shows the current file is `user_repository.py`, the line count is 19, the character count is 36, and the encoding is UTF-8.

Laborator 5:

Cerere PUT pentru a adauga un artist

The screenshot shows a Java development environment with the following components:

- Project View:** Shows the project structure for "Exemplulabot" with files like ContentController.java, ContentService.java, and various DAO and Entity classes.
- Database Tool:** Displays MySQL tables "artists" and "songs".
- Postman Client:** An open browser window titled "ExemplulabotApplication" showing a POST request to "http://localhost:8080/api/songcollection/artists/". The request body is JSON:

```
PUT http://localhost:8080/api/songcollection/artists/
{
  "name": "Disturbed",
  "active": true
}
```

The response shows a 200 OK status with a JSON body:

```
{
  "id": "11a8011a-2f70-4484-b020-ba3237392429",
  "name": "Disturbed",
  "active": true
}
```

At the bottom, the system log indicates:

```
ExemplulabotApplication in 6.712 seconds (JVM running for 7.414)
  Using Spring DispatcherServlet 'dispatcherServlet'
  Using Servlet 'dispatcherServlet'
  Ed initialization in 2 ms
```

Cerere GET pentru a primi datele unui artist identificat cu un uuid

The screenshot shows a Java development environment with the following components:

- Project View:** Shows the project structure for "Exemplulabot" with files like ContentController.java, ContentService.java, and various DAO and Entity classes.
- Database Tool:** Displays MySQL tables "artists" and "songs".
- Postman Client:** An open browser window titled "ExemplulabotApplication" showing a GET request to "http://localhost:8080/api/songcollection/artists/870b3977-6b02-4700-8c41-38e51d4524a". The request has a "Query Params" section:

KEY	VALUE	DESCRIPTION
Key	Value	Description

The response shows a 200 OK status with a JSON body:

```
{
  "id": "870b3977-6b02-4700-8c41-38e51d4524a",
  "name": "Disturbed",
  "active": true
}
```

At the bottom, the system log indicates:

```
ExemplulabotApplication in 6.712 seconds (JVM running for 7.414)
  Using Spring DispatcherServlet 'dispatcherServlet'
  Using Servlet 'dispatcherServlet'
  Ed initialization in 2 ms
```

Cerere GET pentru a primi datele unei melodii identificata cu un id

The screenshot shows a developer's environment with several open windows and tabs:

- File Edit View Navigate Code Refactor Build Run Tools VCS Window Help**
- ExemplLaborator - ExemplLaboratorApplication** (Active tab)
- ContentController.java**: A Java code editor showing a controller class with methods like `getArtistById`, `addArtist`, `getSongCollection`, `updateArtist`, and `updateContent`.
- Scratch Pad**: A tool for testing API endpoints.
- Postman**: A UI for making HTTP requests. The current request is a GET to `http://localhost:8080/api/songcollection/songs/1`. Headers include `Content-Type: application/json`. The response body is a JSON object representing a song:

```
1 | { "id": 1, "name": "Animals", "gense": "blues", "year": 2054, "type": "song"}  
2 |  
3 |  
4 |  
5 |  
6 |  
7 |
```

- Database**: A SQL client showing two tables:
 - ArtistSongs**: Contains rows like `1 Animals blues 2054` and `15 Sels blues 2054`.
 - Artists**: Contains rows like `1 Animals blues 2054` and `15 Sels blues 2054`.
- Console**: Shows command-line output for database queries.
- Project Explorer**: Shows the project structure with packages like `com.lab` and `com.lab.repository`.
- Properties**: Shows application properties like `spring.datasource.url=jdbc:mysql://127.0.0.1:3306/exemplaradb`.
- Run**: Shows the application running on port 8080.
- Output**: Shows build logs.
- Search Bar**: Type here to search.
- System Tray**: Shows battery level, network, and system status.

Cerere PUT pentru a crea o melodie si a o asocia cu un artist

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ExemplarLaboratory ContentController.java

ExemplarLaboratory src main java com labpos ExemplarLaboratory Controller ContentController getContentsByNamePartial

Project src main java com.labpos.ExemplarLaboratory IArtistRepository.java

IArtistRepository.java

no usages

@RequestMapping("api/artist")

public Response

Integer id

String name

String genre

String year

String type

GET get artist by id

PUT add song to artist via id

ArtistController

ArtistController

Collection

pos_project

pos_noSQL

pos_project

GET get artist by id

PUT add artist

Artist

Content

Artist

Content

Artist

Content

MySQL [22] [closed] DB server [SQL, Planning] - Oracle VM VirtualBox

13 Looking for the summer pop hardrock 1969 song

14 No More pop hardrock 2018 song

9 rows in set (0.001 sec)

Marinadb [Artists_Songs]> select * from content_artists;

id	content	id_artist
17	97b00a28-88af-4427-b4ef-c553813c616	
18	97b00a28-88af-4427-b4ef-c553813c616	
19	c340bb04-a3d3-4667-8c7a-37f1086c05d0	
20	c340bb04-a3d3-4667-8c7a-37f1086c05d0	
21	c340bb04-a3d3-4667-8c7a-37f1086c05d0	
22	f1fb011a-2fd7-4484-b020-ba3237392429	
23	f1fb011a-2fd7-4484-b020-ba3237392429	
24	f7065977-b602-4700-8c41-38e51dd4524a	

8 rows in set (0.001 sec)

Marinadb [Artists_Songs]> select * from content;

id	name	genre	year	type
1	Animals	blues	2014	song
17	Sebi	blues	2014	song
18	Tremor	electronic	2014	song
19	Looking (Guey)	hardrock	2001	song
20	ETED	hardrock	2001	song
21	Revenja	hardrock	2005	song
23	Josephine	pop	1965	song
24	Looking for the summer	hardrock	2018	song

23 rows in set (0.000 sec)

Marinadb [Artists_Songs]>

Postman

Scratch Pad

Home Workspaces Explore

Search Postman

GET get artist by id

GET get content by id

PUT add song to artist via id

pos project / add song to artist via uid

PUT http://localhost:8080/api/songcollection/artists/870e3977-b602-4700-8c41-38e51dd4524a/songs

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Send Cookies Beautify

1

2 .. {"name": "No Moze",

3 "genre": "hardrock",

4 "year": 2010,

5 "type": "song"

6

Body Cookies Headers (4) Test Results

200 OK 53 ms 123 B Save Response

Pretty Raw Preview Visualize Text

1

or: 1205-01000: Data truncated for column 'genre' at row 1

Error: 1205, SQLState: 01000

nn=08 Data truncated for column 'genre' at row 1

000010: On release of batch it still contained JDBC statements

Runner Tools

Type here to search

40°F Cloudy 12/20/2023

Cerere GET pentru a gasi artistii dupa nume (potrivire parciala)

The screenshot shows a developer's workspace with several windows open:

- IntelliJ IDEA:** Shows the project structure for `Exemplulaborator` with Java files like `ArtistRepository.java` and `ArtistService.java`.
- Terminal:** Shows MySQL queries running against a MariaDB database. One query is:


```
MariaDB [Artists_Songs]> select * from artists;
```

id	name	genre	year	type
1	Animals	blues	2014	song
1	Seb1	blues	2014	song
18	System	electronic	2014	song
19	Chop Suey!	hardrock	2001	song
20	ATM	hardrock	2001	song
21	One Step	hardrock	2000	song
22	Josephine	pop	1995	single
23	Looking for the summer	pop	1989	song
24	No More	hardrock	2018	song
- Postman:** Shows a GET request to `http://localhost:8080/api/songcollection/artists/?name=r`. The response body is:


```
[{"id": 1, "name": "Chris Rea", "active": true}, {"id": 9, "name": "Disturbed", "active": true}, {"id": 11, "name": "Martin Garrix", "active": false}]
```
- MySQL Workbench:** Shows the same database structure and data as the terminal.
- System Tray:** Shows the date and time as 8:28 PM on 1/22/2023.

Cerere POST pentru a actualiza un artist (nume sau/si daca este activ sau nu)

The screenshot shows a developer's workspace with several windows open:

- IntelliJ IDEA:** Shows the project structure for `Exemplulaborator` with Java files like `ArtistRepository.java` and `ArtistService.java`.
- Terminal:** Shows MySQL queries running against a MariaDB database. One query is:


```
MariaDB [Artists_Songs]> select * from artists;
```

id	name	genre	year	type
1	Chris Rea	System of a Down	1	song
1	Chris Rea	Disturbed	1	song
1	Martin Garrix	Disturbed	1	song
- Postman:** Shows a POST request to `http://localhost:8080/api/songcollection/artists/97bd0a28-88af-4427-b4ef-c5353813c616?name=Martin&active=false`. The response body is:


```
{"id": 9, "name": "Martin Garrix", "active": false}
```
- MySQL Workbench:** Shows the same database structure and data as the terminal.
- System Tray:** Shows the date and time as 8:30 PM on 1/22/2023.

Cerere POST pentru a actualiza o melodie (nume sau/si anul aparitiei sau/si genul sau/si tipul)

The screenshot shows a developer's workspace with several windows open:

- Code Editor:** Displays Java code for `ArtistService.java` and `ContentController.java`. The `ContentController` has a method annotated with `@PUT` and `@Path("/{id}")`, which handles updating a song's details.
- Terminal:** Shows MySQL queries running against a `MariaDB [Artists_Songs]` database. One query filters songs by genre ('electronic') and another retrieves all songs.
- Postman:** An API testing tool window. It shows a `POST` request to `http://localhost:8080/api/songcollection/songs/{id}/genre=electronic`. The `Params` tab includes a checked `genre` parameter set to `electronic`. The `Body` tab shows a JSON payload with fields `name` (set to `Animals`) and `genre` (set to `electronic`).
- System Tray:** Shows system status including battery level (40%), weather (Cloudy), and system time (8:33 PM, 1/22/2023).

Cerere GET pentru afisare paginata si pentru a filtra melodii in functie de nume (parial sau exact) sau/si in functie de anul aparitiei sau/si in functie de genul sau/si in functie de tipul acestaia

This screenshot is similar to the previous one but focuses on a paginated and filtered search:

- Code Editor:** Same Java code as before.
- Terminal:** Same MySQL queries for filtering and retrieving all songs.
- Postman:** A `GET` request to `http://localhost:8080/api/songcollection/songs?page=2`. The `Params` tab includes a checked `page` parameter set to `2`. The `Body` tab shows a JSON response containing two pages of song data, with the second page starting from index 2.
- System Tray:** Same system status as the first screenshot.

ArtisService.java

```

private IContentService contentService;

@RequestMapping(value = "/")
public ResponseEntity<ContentResponse> ContentResponse(@RequestBody ContentRequest content) {
    if(response != null) {
        return ResponseEntity.ok();
    }
    return ResponseEntity.ok();
}

@RequestMapping(value = "/{id}")
public ResponseEntity<ContentResponse> getSong(@PathVariable("id") UUID id) {
    ContentResponse response = contentService.getSong(id);
    if(response != null) {
        return ResponseEntity.ok(response);
    }
    return ResponseEntity.notFound();
}

@RequestMapping(value = "/addSong")
public ResponseEntity<ContentResponse> addSong(@RequestBody ContentRequest content) {
    ContentResponse response = contentService.addSong(content);
    if(response != null) {
        return ResponseEntity.ok(response);
    }
    return ResponseEntity.badRequest();
}

@RequestMapping(value = "/updateSong")
public ResponseEntity<ContentResponse> updateSong(@RequestBody ContentRequest content) {
    ContentResponse response = contentService.updateSong(content);
    if(response != null) {
        return ResponseEntity.ok(response);
    }
    return ResponseEntity.badRequest();
}

```

Postman Test:

```

GET http://localhost:8080/api/songcollection/songs?page=1&items_per_page=4

```

Query Params:

KEY	VALUE	DESCRIPTION
name	r	
genre	electronic	
items_per_page	4	
page	1	

Body (Pretty):

```

[{"id": 1, "name": "Tremor", "genre": "electronic", "year": 2014, "type": "song"}, {"id": 2, "name": "Revolving Girl", "genre": "blues", "year": 2014, "type": "song"}, {"id": 3, "name": "Chop Suey!", "genre": "hardrock", "year": 2001, "type": "song"}, {"id": 4, "name": "No Holes", "genre": "hardrock", "year": 2008, "type": "song"}, {"id": 5, "name": "Animals", "genre": "blues", "year": 2014, "type": "song"}, {"id": 6, "name": "Revenga", "genre": "hardrock", "year": 2005, "type": "song"}, {"id": 7, "name": "Josephine", "genre": "pop", "year": 1985, "type": "song"}, {"id": 8, "name": "Looking for the summer", "genre": "pop", "year": 1989, "type": "song"}, {"id": 9, "name": "Tremor", "genre": "hardrock", "year": 2010, "type": "song"}, {"id": 10, "name": "Sebi", "genre": "blues", "year": 2014, "type": "song"}, {"id": 11, "name": "No More", "genre": "hardrock", "year": 2018, "type": "song"}, {"id": 12, "name": "Hazardous", "genre": "hardrock", "year": 2001, "type": "song"}, {"id": 13, "name": "Chop Suey!", "genre": "hardrock", "year": 2005, "type": "song"}, {"id": 14, "name": "Revolving Girl", "genre": "hardrock", "year": 2008, "type": "song"}, {"id": 15, "name": "Animals", "genre": "blues", "year": 2014, "type": "song"}, {"id": 16, "name": "Revenga", "genre": "hardrock", "year": 2005, "type": "song"}, {"id": 17, "name": "Josephine", "genre": "pop", "year": 1985, "type": "song"}, {"id": 18, "name": "Looking for the summer", "genre": "pop", "year": 1989, "type": "song"}, {"id": 19, "name": "Tremor", "genre": "hardrock", "year": 2010, "type": "song"}, {"id": 20, "name": "Sebi", "genre": "blues", "year": 2014, "type": "song"}, {"id": 21, "name": "No More", "genre": "hardrock", "year": 2018, "type": "song"}, {"id": 22, "name": "Hazardous", "genre": "hardrock", "year": 2001, "type": "song"}, {"id": 23, "name": "Chop Suey!", "genre": "hardrock", "year": 2005, "type": "song"}, {"id": 24, "name": "Revolving Girl", "genre": "hardrock", "year": 2008, "type": "song"}]

```

ArtisService.java

```

private IContentService contentService;

@RequestMapping(value = "/")
public ResponseEntity<ContentResponse> ContentResponse(@RequestBody ContentRequest content) {
    if(response != null) {
        return ResponseEntity.ok();
    }
    return ResponseEntity.ok();
}

@RequestMapping(value = "/{id}")
public ResponseEntity<ContentResponse> getSong(@PathVariable("id") UUID id) {
    ContentResponse response = contentService.getSong(id);
    if(response != null) {
        return ResponseEntity.ok(response);
    }
    return ResponseEntity.notFound();
}

@RequestMapping(value = "/addSong")
public ResponseEntity<ContentResponse> addSong(@RequestBody ContentRequest content) {
    ContentResponse response = contentService.addSong(content);
    if(response != null) {
        return ResponseEntity.ok(response);
    }
    return ResponseEntity.badRequest();
}

@RequestMapping(value = "/updateSong")
public ResponseEntity<ContentResponse> updateSong(@RequestBody ContentRequest content) {
    ContentResponse response = contentService.updateSong(content);
    if(response != null) {
        return ResponseEntity.ok(response);
    }
    return ResponseEntity.badRequest();
}

```

Postman Test:

```

GET http://localhost:8080/api/songcollection/songs?name=r&items_per_page=4&page=1

```

Query Params:

KEY	VALUE	DESCRIPTION
name	r	
genre	electronic	
items_per_page	4	
page	1	

Body (Pretty):

```

[{"id": 1, "name": "Tremor", "genre": "electronic", "year": 2014, "type": "song"}, {"id": 2, "name": "Revolving Girl", "genre": "blues", "year": 2014, "type": "song"}, {"id": 3, "name": "Chop Suey!", "genre": "hardrock", "year": 2001, "type": "song"}, {"id": 4, "name": "No Holes", "genre": "hardrock", "year": 2008, "type": "song"}, {"id": 5, "name": "Animals", "genre": "blues", "year": 2014, "type": "song"}, {"id": 6, "name": "Revenga", "genre": "hardrock", "year": 2005, "type": "song"}, {"id": 7, "name": "Josephine", "genre": "pop", "year": 1985, "type": "song"}, {"id": 8, "name": "Looking for the summer", "genre": "pop", "year": 1989, "type": "song"}, {"id": 9, "name": "Tremor", "genre": "hardrock", "year": 2010, "type": "song"}, {"id": 10, "name": "Sebi", "genre": "blues", "year": 2014, "type": "song"}, {"id": 11, "name": "No More", "genre": "hardrock", "year": 2018, "type": "song"}, {"id": 12, "name": "Hazardous", "genre": "hardrock", "year": 2001, "type": "song"}, {"id": 13, "name": "Chop Suey!", "genre": "hardrock", "year": 2005, "type": "song"}, {"id": 14, "name": "Revolving Girl", "genre": "hardrock", "year": 2008, "type": "song"}, {"id": 15, "name": "Animals", "genre": "blues", "year": 2014, "type": "song"}, {"id": 16, "name": "Revenga", "genre": "hardrock", "year": 2005, "type": "song"}, {"id": 17, "name": "Josephine", "genre": "pop", "year": 1985, "type": "song"}, {"id": 18, "name": "Looking for the summer", "genre": "pop", "year": 1989, "type": "song"}, {"id": 19, "name": "Tremor", "genre": "hardrock", "year": 2010, "type": "song"}, {"id": 20, "name": "Sebi", "genre": "blues", "year": 2014, "type": "song"}, {"id": 21, "name": "No More", "genre": "hardrock", "year": 2018, "type": "song"}, {"id": 22, "name": "Hazardous", "genre": "hardrock", "year": 2001, "type": "song"}, {"id": 23, "name": "Chop Suey!", "genre": "hardrock", "year": 2005, "type": "song"}, {"id": 24, "name": "Revolving Girl", "genre": "hardrock", "year": 2008, "type": "song"}]

```

Screenshot of the Eclipse IDE interface showing the Exemplulaborator project structure and code editor. The code editor displays `ArtistService.java` with annotations for `@RequestMapping` and `@ContentResponseEntity`. Below the code is a MySQL database query window showing results from the `Artists_Songs` table.

Postman API testing tool is open, showing a GET request to `http://localhost:8080/api/songcollection/songs/?name=ATWA&match=1`. The request includes query parameters: `genre: electronic`, `match: 1`, and `name: ATWA`. The response body is a JSON object:

```

1  {
2    "id": 26,
3    "name": "ATWA",
4    "genre": "hardrock",
5    "year": 2001,
6    "type": "song"
7  }
8
9

```

Screenshot of the Eclipse IDE interface showing the same project structure and code editor. The code editor displays `ArtistService.java`. Below the code is a MySQL database query window showing results from the `Artists_Songs` table.

Postman API testing tool is open, showing a GET request to `http://localhost:8080/api/songcollection/songs/?genre=electronic`. The request includes query parameters: `genre: electronic` and `name: ATWA`. The response body is a JSON array:

```

1  [
2    {
3      "id": 1,
4      "name": "Animals",
5      "genre": "electronic",
6      "year": 2014,
7      "type": "song"
8    },
9    {
10      "id": 18,
11      "name": "Tremor",
12      "genre": "electronic",
13      "year": 2014,
14      "type": "song"
15    }
16  ]

```

Laborator 6:

Cerere GET pentru a afisa toate playlisturile

The screenshot shows a developer's environment with multiple windows open. On the left is the IntelliJ IDEA interface, displaying Java code for a 'PlaylistController' class. The code includes annotations like @RequestMapping and @ResponseBody, and methods for creating playlists and getting songs. In the center is a browser window for Postman, which is making a GET request to 'http://localhost:8080/api/playlist/'. The response body is a JSON object representing a playlist with an ID, title, and a list of songs. On the right is a terminal window showing a log of HTTP requests and responses, likely from a JMeter or similar tool, with logs such as 'INFO 12352 --- [main]' and various status codes and timestamps. The bottom of the screen features the Windows taskbar with icons for Start, File Explorer, Task View, and several pinned applications.

Cerere GET pentru a afisa toate playlisturile unui user

The screenshot displays a dual-pane development environment. On the left, the IntelliJ IDEA interface shows a project structure for a 'playlists' application. The 'src/main/java/com/pos/playlist/controller' package contains a 'PlaylistController.java' file with code for handling playlist requests. On the right, the Postman application is open, showing a 'Collections' section with a 'paw_project' collection containing a 'pos noSQL' folder. A specific API endpoint, 'GET http://localhost:8080/api/playlist/1', is selected in Postman's 'Scratch Pad'. The request includes parameters for 'id' (set to '1') and 'userId' (set to '1'). The response body is displayed as JSON, representing a playlist entity with an ID, title, and songs.

```
1 {
2   "id": "63cc1d0482bd64271b8808a2",
3   "title": "Playlist1",
4   "songs": [
5     {
6       "id": 1,
7       "title": "song1",
8       "href": "link1"
9     },
10    {
11      "id": 2,
12      "title": "song2",
13      "href": "link2"
14    }
15  ],
16  "userId": 1
17},
18{
19  "id": "63cc1d0482bd64271b8808a3",
20  "title": "Playlist2",
21  "songs": []
22}
```

Cerere DELETE pentru a sterge o anumita melodie dintr-un playlist

```

    @RequestMapping(value = "/{songId}", method = RequestMethod.DELETE)
    public ResponseEntity<HttpStatus> deleteSongFromPlaylist(@PathVariable Long songId) {
        try {
            playlistService.deleteSongFromPlaylist(songId);
            return ResponseEntity.ok().build();
        } catch (UserDoesn'tExistException e) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }

```

```

    @GetMapping("/?userId={userId}")
    public ResponseEntity<List<Playlist>> getAllPlaylistsFromUser(@PathVariable Long userId) {
        try {
            List<Playlist> playlists = playlistService.getAllPlaylistsFromUser(userId);
            return ResponseEntity.ok(playlists).build();
        } catch (UserDoesn'tExistException e) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }

```

Cerere DELETE pentru a sterge un intreg playlist

The screenshot shows an IDE interface with a Java project named "playlists". The code editor displays `PlaylistController.java` which contains a `DELETE` method for deleting a playlist by user ID. The Postman tab shows a `DELETE` request to `http://localhost:8080/api/playlist/63cc106c2b6e64271b05b0a2?userId=1` with a query parameter `userId` set to `1`. The response body is `1`.

The screenshot shows an IDE interface with a Java project named "playlists". The code editor displays `PlaylistController.java` which contains a `GET` method for getting all playlists from a user. The Postman tab shows a `GET` request to `http://localhost:8080/api/playlist/?userId=1` with a query parameter `userId` set to `1`. The response body is a JSON object:

```

1: {
2:   "id": "63cc106c2b6e64271b05b0a3",
3:   "title": "Playlist2",
4:   "songs": [],
5:   "userId": 1
}
    
```

Cerere POST pentru a adauga melodii la un playlist

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "playlists [playlist]" and contains packages like com.pos.playlist, com.pos.playlist.controller, com.pos.playlist.entity, com.pos.playlist.exception, and com.pos.playlist.repository.
- Code Editor:** The file "PlaylistController.java" is open, showing a POST method for adding a song to a playlist. The code handles various exceptions and returns a ResponseEntity<Song>.
- Terminal:** Shows log output from a run of the application, indicating successful startup and several INFO logs related to the application's internal state.
- Postman:** An API test is being run against the endpoint `http://localhost:8080/api/playlist/63cc106c2b6e64271b05b0a3/song?userId=1`. The request body is JSON, containing an "id": 2 and a "userId": 1. The response is a 200 OK status with a response time of 15 ms and a size of 123 B.

This screenshot shows the same IntelliJ IDEA environment and terminal log as the previous one, but with a different Postman test:

- Postman:** An API test is being run against the endpoint `http://localhost:8080/api/playlist/?`. The response is a 200 OK status with a response time of 11 ms and a size of 280 B. The response body is a JSON array containing a single playlist object.

```

[{"id": "63cc106c2b6e64271b05b0a3", "title": "Playlist2", "songs": [{"id": 2, "title": "song2", "userId": "1"}], "userId": 1}
  
```

Cerere PUT pentru a crea un nou playlist

The screenshot shows the IntelliJ IDEA interface with a Java project named "playlists". The code editor displays `PlaylistController.java` containing a `createPlaylist` method. The `Body` tab in Postman shows the JSON payload:

```

1 {
2   "title": "Playlist3",
3   "userId": 3
4 }

```

The Postman response shows a successful `200 OK` status.

The screenshot shows the IntelliJ IDEA interface with the same Java project. The code editor displays `PlaylistController.java`. The `Body` tab in Postman shows the JSON response for a `GET /playlist` request:

```

1 [
2   {
3     "id": "63cc108c2bde64271b89b8a3",
4     "title": "Playlist2",
5     "songs": [
6       {
7         "id": 2,
8         "title": "song2",
9         "href": "link2"
10       }
11     ],
12     "userId": 1
13   },
14   {
15     "id": "63cc107b2bde64271b89b8a4",
16     "title": "Playlist1",
17     "songs": [],
18     "userId": 2
19   },
20   {
21     "id": "63cd991d8e0ce779e7e818f9b",
22     "title": "Playlist3",
23     "songs": [],
24     "userId": 3
25   }
26 ]

```

The Postman response shows a successful `200 OK` status.