Database Interface - Python implementation

Zotec Alexandru, 432F

TV Management System is a desktop application designed to provide an intuitive interface for interacting with a relational database. The system enables CRUD (Create, Read, Update, Delete) operations for managing clients, TV channels, and their relationships.

# Introduction

Built with Flask for backend functionality and MySQL for data storage, the project demonstrates effective web development practices for database management. It is structured for ease of use, making it accessible to a wide range of users, including those with limited technical expertise. The interface uses HTML and CSS for a clean and straightforward design, enabling seamless navigation and interaction.

# IMPLEMENTATION

The application is structured around Flask for backend routing, MySQL for data storage, and HTML/CSS for user interface design. CRUD functionalities are implemented via Flask routes, which connect the frontend interface with the database logic.

## **Functionalities**

The TV Management System offers the following operations:

1. Clients Management

* View a list of clients.
* Add new clients through a form.
* Modify client details (name and preferences).
* Delete clients and associated relationships.

A screen shot of a computer code

Description automatically generated

Fig1 – View Clients

1. TV Channels Management

* View a list of TV channels.
* Add new TV channels with a title and channel number.
* Modify TV channel details.
* Delete TV channels and associated relationships.
* A screen shot of a computer program

  Description automatically generatedFig2 – View TV Channels

1. Relationships Management

* View relationships between clients and TV channels.
* Create new relationships by selecting a client and a TV channel.
* Delete relationships.
* A screen shot of a computer code

  Description automatically generated

Fig3 – View Relationships

## Routes

Flask routes handle HTTP requests and map them to backend logic. They serve as the backbone of the application, enabling integration between the frontend and the database.

For example:

* The route /clients displays a list of all clients and handles adding, modifying, and deleting clients.
* The route /tv\_channels allows users to manage TV channels.
* The route /tv\_clients displays relationships between clients and TV channels.

Below is an example of a route:

A screen shot of a computer program

Description automatically generated

Fig 4. Route

## Templates

The templates folder contains HTML files that define the application's user interface. Flask dynamically renders these templates in response to HTTP requests.

Key templates include:

1. **index.html**

* Displays the homepage.
* A screen shot of a computer program

  Description automatically generated

1. **clients.html**

* Displays a list of clients and provides forms for adding, modifying, and deleting clients.
* A computer screen shot of text

  Description automatically generated
* Fig 5 – Clients.html

1. **tv\_channels.html**

* Displays a list of TV channels with forms for adding, modifying, and deleting channels.

1. **tv\_clients.html**

* Displays relationships between clients and TV channels.

**D. Database Integration** The application uses MySQL as the database backend. Connections are established using MySql.Data.MySqlClient, and queries are executed to perform CRUD operations. The database contains three tables: Clients, TVChannels, and TV\_Clients, representing clients, TV channels, and their relationships, respectively

A computer screen shot of text

Description automatically generated

Fig 6 – Database Connection

# erd (entity relationship diagram)

A close-up of a label

Description automatically generated

# results

The following tests verify the application's functionalities:

1. Adding a Client

* Input: Name = "John Doe", Preferences = "News".
* Steps: Fill in the form and click "Add Client".
* Expected: "John Doe" appears in the client list.
* Actual: Success.

1. Adding a TV Channel

* Input: Title = "HBO", Channel No = 5.
* Steps: Fill in the form and click "Add Channel".
* Expected: "HBO" appears in the channel list.
* Actual: Success.

1. Creating a Relationship

* Input: Client = "John Doe", Channel = "HBO".
* Steps: Select a client and a channel, then click "Create Relationship".
* Expected: Relationship is displayed.
* Actual: Success.

##### V.UX(USER EXPERIENCE)

**The application uses a simple styles.css file to enhance the UI:**

A screen shot of a computer program

Description automatically generated

Fig 7 – Css file

A screenshot of a computer

Description automatically generated

Fig8 - Main Page

A screenshot of a computer

Description automatically generated

Fig9 - Manage Clients

A screenshot of a tv channel

Description automatically generated

Fig10 - Manage TV Channels

A screenshot of a computer

Description automatically generated

Fig11 - Manage Relationships

# vi.conclusion

The TV Management System successfully demonstrates CRUD operations on a MySQL database through a Flask web application. The interface is user-friendly and provides essential functionalities such as adding, viewing, editing, and deleting data. Users can seamlessly manage clients, TV channels, and their relationships through a web-based interface, simplifying database interactions without requiring direct SQL knowledge.

The project effectively integrates Flask as the backend framework, MySQL as the relational database, and Jinja2 for dynamic HTML rendering. By structuring database interactions through Flask routes, the system ensures clear separation of concerns and maintains a modular architecture. Each CRUD operation is efficiently mapped to specific routes, making the application highly maintainable and extensible.

Future improvements could enhance security features such as user authentication and authorization, preventing unauthorized modifications. Additionally, implementing search and filter functionalities would improve usability, allowing users to quickly navigate large datasets. Enhancing the UI with modern styling using CSS frameworks like Bootstrap or Materialize would further improve the user experience.

Overall, the TV Management System serves as a practical demonstration of database-driven web application development, showcasing how Flask can efficiently interact with MySQL while providing a structured and scalable approach to database management.

VII.REFERENCES

1. Flask Documentation, "Flask Web Development Framework," 2023. [Online]. Available: <https://flask.palletsprojects.com/>.

2. MySQL Documentation, "MySQL 8.0 Reference Manual," 2023. [Online]. Available: <https://dev.mysql.com/doc/>.

3. Jinja Documentation, "Jinja Templating for Flask," 2023. [Online]. Available: https://jinja.palletsprojects.com/.

4. W3Schools, "HTML Forms and Inputs," 2023. [Online]. Available: https://www.w3schools.com/html/html\_forms.asp.

5. W3Schools, "CSS Styling Guide," 2023. [Online]. Available: https://www.w3schools.com/css/.

6. Python Software Foundation, "Python 3 Documentation," 2023. [Online]. Available: <https://docs.python.org/3/>.

7. Flask-SQLAlchemy Extension, "Flask-SQLAlchemy Documentation," 2023. [Online]. Available: <https://flask-sqlalchemy.palletsprojects.com/>.

8. Bootstrap Documentation, "Bootstrap CSS Framework," 2023. [Online]. Available: https://getbootstrap.com/docs/5.3/getting-started/introduction/.

9. Mozilla Developer Network (MDN), "JavaScript and DOM Manipulation," 2023. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.

10. IEEE Citation Standards, "How to Cite Sources in IEEE Format," 2023. [Online]. Available: <https://ieeeauthorcenter.ieee.org/>.

11. Flask-WTF Extension, "Flask-WTF Forms," 2023. [Online]. Available: <https://flask-wtf.readthedocs.io/en/stable/>.

12. GitHub, "Flask Example Projects," 2023. [Online]. Available: <https://github.com/topics/flask>.