BTEC
Alliance with FPT Education
BTEC FPT BRITISH COLLEGE

ASSIGNMENT 2

Qualification	**BTEC Level 5 HND Diploma in Computing**		
Unit number and title	Unit 9: Software Development Life Cycle		
Submission date	31/07/2023	**Date Received 1st submission**	
Re-submission Date		**Date Received 2nd submission**	
Student Name	Conghoang.trann@gmail.com	**Student ID**	0862620450
Class		**Assessor name**	

Student declaration

I certify that the assignment submission is entirely my work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice.

Student's signature	Conghoang.tran

Grading grid

P5	P6	P7	M3	M4	M5	M6	D3
✓	✓	✓					

<table>
<tr><td colspan="2">❒ Summative Feedback:</td><td colspan="2">❒ Resubmission Feedback:</td></tr>
<tr><td>Grade:</td><td colspan="2">Assessor Signature:</td><td>Date:</td></tr>
<tr><td colspan="4">Internal Verifier’s Comments:</td></tr>
<tr><td colspan="4">Signature & Date:</td></tr>
</table>

Table of Contents

A. Introduction 5

B. Content 6

P5: Undertake a software investigation to meet a business need. 6

I. TuneSource Project Overview. 6

1) About the Tune Source project and its goals. 6

2) Importance of requirements definition in software development. 6

II. The concept of Requirement. 7

1) Definition of the requirement. 7

2) Different types of requirements. 7

3) Criteria for assessing the importance of requirements. 7

III. Classification of requests. 8

1) Functional requirements. 8

2) Non-functional requirements. 8

3) Difference between functional requirements and non-functional requirements. 9

IV. Applied in Tune Source project. 9

1) List the functional and non-functional requirements for the Tune Source project. 9

2) Detailed description of each requirement and their importance to the project. 10

3) Prioritize requirements and plan implementation. 10

V. Requirements gathering techniques in Tune Source. 11

1) List the functional and non-functional requirements for the Tune Source project. 11

2) Detailed description of each requirement and their importance to the project. 11

3) Prioritize requirements and plan implementation. 12

4) Business Process Improvement Techniques. 13

VI. Conclude. 15

P6: Use appropriate software analysis tools/techniques to carry out a software investigation and create supporting documentation. 16

I. Use Case Diagrams. 16

1) Introduction to Use Case Diagrams. 16

2) Components of a Use Case Diagram. 16

3) Creating a Use Case Diagram. 17
4) Best Practices for Use Case Diagrams. 18
5) Applying Use Case Diagrams to a Project of TuneSource. 18
II. Data flow Diagram. 20
1) Introduction to Data Flow Diagrams. 20
2) Components of a Data Flow Diagram. 21
3) Creating a Data Flow Diagram. 23
4) Best Practices for Data Flow Diagrams. 23
5) Applying Data Flow Diagrams to a project of TuneSource. 24
III. Entity Relation Diagram. 26
1) Introduction to Entity-Relationship Diagrams. 26
2) Components of an Entity-Relationship Diagram. 27
3) Creating an Entity-Relationship Diagram. 28
4) Best Practices for Entity-Relationship Diagrams. 29
5) Applying Entity-Relationship Diagrams to a Project. 29
P7: Explain how user and software requirements have been addressed. 32
I. Wireframe of the project. 32
1. Wireframe for login and registration 32
2. Wireframe for admin page 33
II. Database design. 36
1) Data tables. 36
2) Relationships between tables. 38
3) Architectural design. 38
III. User manual for the project. 39
1) Instructions for registration and login 39
2) Instructions for adding, editing, and deleting artists 40
3) Instructions for adding, editing, and deleting genres 46
4) Instructions for adding, editing and deleting albums 50
5) Instructions for adding, editing and deleting songs 55

6) Logout 60

C. Conclusion. 62

D. References 63

List of Figure

Figure 1: Use case Diagram 16
Figure 2: Use case diagram of online shopping system 17
Figure 3: Use case diagram of user 19
Figure 4: Use case diagram of admin 20
Figure 5: Data flow diagram 21
Figure 6: Components of a data flow diagram 22
Figure 7: Data flow diagram of student management 23
Figure 8: Context Diagram 25
Figure 9: Level 0 Admin 26
Figure 10: Entity Relation Diagram 27
Figure 11: ERD 30
Figure 12: Wireframe login 32
Figure 13: Wireframe register 33
Figure 14: Wireframe dasdboard 33
Figure 15: Wireframe album 34
Figure 16: Wireframe song 34
Figure 17: Wireframe genre 35
Figure 18: Wireframe artist 35
Figure 19: Web register 39
Figure 20: Web login 40
Figure 21: Web dasdhoard 40
Figure 22: Web artist 41
Figure 23: Button Add artist 41
Figure 24: Add artist 42
Figure 25: Action add new arrtist 42
Figure 26: After when add artist success 43
Figure 27: Edit artist 43
Figure 28: Adit Artist 44
Figure 29: Input information need edit 44
Figure 30: After edit artist 45
Figure 31: Delete Artist 45
Figure 32: Delete Success 46
Figure 33: Add genre 46

Figure 34: Click add genre 47
Figure 35: Add genre 47
Figure 36: Edit genre 48
Figure 37: Edit genre 48
Figure 38: Edit genre 49
Figure 39: Edit genre success.....49
Figure 40: Delete genre 50
Figure 41: Delete genre 50
Figure 42: Web album 51
Figure 43: Add Album.....51
Figure 44: Add album 52
Figure 45: After add album.....52
Figure 46: Edit album 53
Figure 47: Edit success.....54
Figure 48: Delete album 54
Figure 49: Add song.....55
Figure 50: Add song.....55
Figure 51: Add song.....56
Figure 52: Input data 56
Figure 53: Save song.....57
Figure 54: Add song success 57
Figure 55: Button edit song 58
Figure 56: Edit web.....58
Figure 57: Edit success.....59
Figure 58: Delete song.....59
Figure 59: Warning delete 60
Figure 60: After delete 60
Figure 61: Logout 61
Figure 62: Logout success.....61

List of Table

Table 1: Table Admin 36
Table 2" Table artist.....36
Table 3: Table genre 37
Table 4: Table album 37
Table 5: Table song.....38

A. Introduction

In the first report, I discussed software lifecycle models for Tune Source, a southern California-based company that specializes in hard-to-find classical music recordings. I evaluated four different software lifecycle models: Waterfall, V-Model, Spiral, and Rapid Development and determined that the Waterfall model could be the best choice for Tuning Sources. We also discuss risk management and the importance of conducting a feasibility study.

In this second report, I will continue to analyze Tune Source's project to make it possible to sell digital music downloads to customers through their in-store kiosks and over the Internet by use their website. The report is divided into several sections, each addressing a specific aspect of the project.

Section P5 discusses the software investigation done to meet the business needs of Tune Source. This involves identifying business needs and gathering information from stakeholders such as customers, employees, and management using techniques such as interviews, surveys, and focus groups. Existing documents and data are also analyzed to identify business needs and requirements.

Section P6 describes the use of appropriate software analysis tools and techniques to perform software investigations and generate supporting documentation. This includes documenting requirements using tools like user stories, use cases, and functional requirements, and creating diagrams like Use Case diagrams, Relationship diagrams entity system (ERD) and Data Flow diagram (DFD) to help visualize and understand the system to be implemented.

Section P7 explains how to address user and software requirements. This involves using methods such as software behavior specification methods to model the behavior of the system and show how it meets requirements. The reliability and efficiency of the software is also evaluated by conducting tests such as unit testing, integration testing, and system testing.

It is my hope that this report will provide a comprehensive and detailed analysis of the Tune Source project, including specific sections that address various aspects of the project.

B. Content

P5: Undertake a software investigation to meet a business need.

I. TuneSource Project Overview.

1) About the Tune Source project and its goals.

a) About the Tune Source project

Tune Source is a company based in southern California, founded by three entrepreneurs with connections to the music industry: John Margolis, Megan Taylor, and Phil Cooper. Initially, John and Phil teamed up to open a number of stores in southern California specializing in classic and hard-to-find jazz, rock, country and folk recordings. Megan was later invited to join the company because of her connections and knowledge of classical music. Tune Source is quickly becoming the place to go for rare audio recordings. Last year's annual sales was $40 million with an annual growth of about 3%-5% per year. Tune Source now has a website that allows customers to search and buy CDs.

b) The goal of P5

The goal of P5 was to conduct a software investigation to meet the business needs of Tune Source. Specifically, this project was initiated to increase sales by creating the ability to sell downloaded digital music to customers through their in-store kiosks and over the Internet using their website. To this end, P5 will focus on defining the requirements for the software and using the appropriate software analysis tools/techniques to perform the software investigation and generate supporting documentation.

2) Importance of requirements definition in software development.

Determining requirements is an important step in the software development process. Requirements are the conditions or capabilities that the software needs to meet in order to meet user needs and achieve business goals. Defining requirements helps ensure that the software will meet user needs and deliver value to the business.

Failure to define requirements correctly can lead to software development that does not meet user needs, resulting in wasted time and costs. In addition, failure to define the exact requirements can also lead to software errors or omissions, making it difficult to use and maintain.

Therefore, defining requirements is an important step to ensure that the software will meet the needs of users and bring value to the business. This saves time and money in the software development process, and increases the likelihood of project success.

II. The concept of Requirement.

1) Definition of the requirement.

Requirements are the conditions or capabilities that a system, product or service needs to meet to meet user needs and achieve business goals. In software development, requirements play an important role in shaping the functionality and features of the software, ensuring that it will meet user needs and deliver value to the business.

There are two main types of requirements, functional requirements and non-functional requirements. Functional requirements describe what the software should do, including specific functions and features. Non-functional requirements describe other factors that affect the operation of the software, including performance, scalability, security, etc.

Determining requirements is an important step in the software development process, helping to ensure that the software will meet the needs of users and bring value to the business.

2) Different types of requirements.

In software development, there are two main types of requirements: functional requirements and non-functional requirements.

Functional requirements describe what the software should do, including specific functions and features. For example, in the Tune Source project, some functional requirements might include allowing users to search music in a digital music archive, listen to music, purchase individual downloads for a fixed fee per download. download, set up a customer subscription account that allows unlimited downloads for a monthly fee, and purchase a music download gift card.

Non-functional requirements describe other factors that affect the operation of the software, including performance, scalability, security, etc. For example, in the Tune Source project, some non-functional requirements might include ensuring that the system can handle high traffic during peak periods, ensuring the security of personal information. customers and ensure the stability and usability of the system.

Both of these types of requirements are important in shaping the functionality and features of the software and ensuring that it will meet the needs of the users and deliver value to the business.

3) Criteria for assessing the importance of requirements.

To assess the importance of requirements, there are several criteria that can be used, including:

- Revenue Impact: Requests can affect a business's revenue, for example new features that can attract more customers or increase sales.

- Deployment difficulty: Requirements can have varying degrees of difficulty in implementation, for example some requirements may take time and resources to develop.
- Impact on customer experience: Requirements can affect the customer's experience when using a product or service, for example new features can make the product easy to use more usable or improve performance.
- Conformity with business strategy: Requirements may be in line with the business strategy of the enterprise, for example, some requirements can help the enterprise expand the market or strengthen the competition.

Using these criteria to assess the importance of requirements can help you prioritize requirements and plan for implementation accordingly.

III. Classification of requests.

1) Functional requirements.

a) Define.

Functional requirements are requirements that describe what the software needs to do, including specific functions and features. These requirements define the behavior of the system as it interacts with users or other systems.

b) For example.

In an online sales application, some functional requirements might include allowing users to search for products, view product details, add products to cart, pay for orders, and place orders. Order status tracking. These requirements describe the specific functions the application needs to meet the user's needs.

In the Tune Source project, some non-functional requirements may include ensuring that the system can handle high traffic during peak periods, ensuring the security of customers' personal information. and ensure the stability and usability of the system.

2) Non-functional requirements.

a) Define.

Non-functional requirements are those that describe the properties or characteristics of the system, not the specific functions it needs to perform. These requirements relate to factors such as performance, scalability, security, usability, and system stability.

b) For example.

In an online sales application, some non-functional requirements may include ensuring that the system can handle high traffic during peak periods, ensuring the security of personal information. customers and ensure the stability and usability of the system. These requirements do not describe

the specific functions the application should have, but do affect the operation and user experience of using the application.

In the Tune Source project, some non-functional requirements may include ensuring that the system can handle high traffic during peak periods, ensuring the security of customers' personal information. and ensure the stability and usability of the system.

3) Difference between functional requirements and non-functional requirements.

Functional requirements and non-functional requirements are the two main types of requirements in software development. Functional requirements describe what the software should do, including specific functions and features. These requirements define the behavior of the system as it interacts with users or other systems. For example, in an e-commerce application, some functional requirements might include allowing users to search for products, view product details, add products to cart, pay for orders. and track order status.

Whereas, non-functional requirements describe the properties or characteristics of the system, not the specific functions it needs to perform. These requirements relate to factors such as performance, scalability, security, usability, and system stability. For example, in an online sales application, some non-functional requirements might include ensuring that the system can handle high traffic during peak periods, ensuring information security. customers' personal information and ensure the stability and usability of the system.

The key difference between functional requirements and non-functional requirements is that functional requirements focus on specific functions and features of the software, while non-functional requirements deal with other factors that affect the operation of the software. soft. Both of these types of requirements are important in shaping the functionality and features of the software and ensuring that it will meet the needs of the users and deliver value to the business.

IV. Applied in Tune Source project.

1) List the functional and non-functional requirements for the Tune Source project.

a) TuneSource project functional requirements.

- ✓ Search music in digital music archive
- ✓ Listen to music
- ✓ Purchase individual downloads for a flat fee per download
- ✓ Set up a customer subscription account that allows unlimited downloads for a monthly fee
- ✓ Buy music download gift cards

b) TuneSource project non-functional requirements.

- ✓ Performance: The system can handle high traffic during peak periods
- ✓ Security: Ensuring the safety of customers' personal information
- ✓ Usability: Ensure system stability and usability.

2) Detailed description of each requirement and their importance to the project.

- ❖ **Search music in digital music archive**: Allows users to search for songs by song title, artist name, genre. This feature is important to make it easy for users to find the music they are looking for.
- ❖ **Listen to music**: Allows users to listen to a short sample of the song before deciding to buy. This feature helps users have more information to make purchasing decisions.
- ❖ **Buy individual downloads for a flat fee per download**: Allows users to purchase individual songs for a flat fee. This feature gives users more choices when making purchases and helps businesses get revenue from retail.
- ❖ **Set up a customer subscription account that allows unlimited downloads for a monthly fee**: Allows users to sign up for an account for unlimited downloads for a monthly fee. This feature helps businesses get recurring revenue from customers and helps customers save costs when shopping.
- ❖ **Buy music download gift cards**: Allows users to purchase gift cards to give to others, allowing recipients to use the card to purchase music on the website. This feature helps businesses attract new customers and help customers have more choices of gifts for their loved ones.
- ❖ **Performance**: The system can handle high traffic during peak periods. This requirement is very important to ensure that the system works stably and is not overloaded when there are many people accessing it at the same time.
- ❖ **Security**: Ensuring the safety of customers' personal information. This requirement is important to protect the privacy of your customers and to uphold their trust in the business.
- ❖ **Usability**: Ensure the stability and usability of the system. This requirement is very important to make it easy for users to use the system and have a good shopping experience.

3) Prioritize requirements and plan implementation.

To prioritize requirements and plan for deployment, you can take the following steps:

- **Determine the priority of requirements**: Evaluate the importance of each requirement based on criteria such as revenue impact, difficulty in implementation, impact level customer experience and alignment with business strategy. Based on this assessment, determine a priority for each requirement.

- **Deployment planning**: Based on the priority of the requirements, plan the implementation to suit the available resources and time. Define an estimated time to complete each requirement and create a detailed implementation schedule.
- **Follow the plan**: Implement the requirements according to the planned schedule and monitor the progress to ensure that the project is completed on time. If there are changes in the implementation process, update the plan and notify stakeholders.

Prioritizing requirements and planning implementation carefully will help you make good use of the resources and time available to complete the project efficiently.

V. Requirements gathering techniques in Tune Source.

1) List the functional and non-functional requirements for the Tune Source project.

There are various techniques for requirements gathering in software development, including interviews, surveys, focus groups, and document and data analysis.

- **Interviewing**: Interviewing is a requirement gathering technique through direct communication with stakeholders, such as users, customers or sales staff. Interviews help gather detailed information about the needs and wants of stakeholders and help define requirements for the software.
- **Survey**: Survey is a inquiry gathering technique through sending questions to a large group of people to gather information about their needs and desires. Surveys help gather information from a representative group and help identify common requirements for the software.
- **Focus group**: Focus group is a requirements gathering technique through holding group discussions with stakeholders to gather information about their needs and wants. Focus groups help gather insights from a variety of sources and help define requirements for the software.
- **Document and data analysis**: Document and data analysis is a requirements gathering technique through the analysis of available documents and data to determine requirements for software. This technique helps to define requirements based on already available information and saves time and cost in requirements-gathering process.

All of these techniques have their own pros and cons and can be used flexibly according to the needs of the project.

2) Detailed description of each requirement and their importance to the project.

Requirements gathering techniques that can be applied to a Tune Source project are as follows:

- **Interviews**: Software developers may interview stakeholders, such as users, customers, or salespeople, to gather information about their needs and desires for the software. The questions in the interview can be prepared in advance and focus on topics related to the project.
- **Survey**: Software developers can design a survey and send it to a large group of users to gather information about their needs and desires for the software. Survey questions can be designed to gather specific information about software features and functions.
- **Focus groups**: Software developers can hold group discussions with stakeholders to gather information about their needs and desires for the software. During these discussions, stakeholders can exchange ideas and make suggestions about software features and functions.
- **Document and data analysis**: Software developers can analyze existing documents and data, such as business reports or customer data, to determine requirements for the software. This analysis helps to identify requirements based on the information already available and saves time and cost in the requirements collection process.

3) Prioritize requirements and plan implementation.

Prioritizing requirements and planning for implementation is an important step in the software development process. It involves determining the relative importance of the different requirements and deciding the order in which to execute them.

To prioritize requirements, several factors can be considered, such as the importance of the requirement to the success of the project, the cost and effort required to implement it, and the dependency between different requirements. Various techniques can be used to prioritize requirements, such as the MoSCoW method, where the requirements are classified as Must-Have, Should-Have, May-Have or Won't.

Once the requirements have been prioritized, an implementation plan can be developed. This involves deciding in which order the requirements will be fulfilled, assigning resources and responsibilities, and establishing timelines for completion. The implementation plan should be regularly reviewed and updated as necessary to ensure that the project stays on track.

In summary, prioritizing requirements and planning their implementation is an important step to ensure that the most important requirements are addressed first and that the project is completed on time and within scope. micro-budget. It involves careful consideration of various factors and the use of appropriate techniques to determine the relative importance of different requirements. A well-developed execution plan can help ensure that the project is completed successfully.

Here are some additional details on each step:

- **Prioritizing Requirements**: This step involves evaluating each requirement to determine its relative importance to the success of the project. Factors that can be considered include the value the requirement brings to the project, its impact on other requirements, and its feasibility in terms of cost and effort. Techniques such as the MoSCoW approach can be used to classify requirements into different categories based on their importance.
- **Implementation planning**: Once the requirements have been prioritized, an implementation plan can be developed. This involves deciding in which order the requirements will be fulfilled, assigning resources and responsibilities, and establishing timelines for completion. The deployment plan should account for dependencies between different requirements and should be flexible enough to accommodate changes as needed.
- **Review and update**: The implementation plan should be regularly reviewed and updated as necessary to ensure that it remains in line with the project's goals and objectives. This may involve changing the order in which requirements are fulfilled or adjusting timelines to accommodate new developments.

Overall, prioritizing requirements and planning their implementation is an important step in ensuring that a software development project is completed successfully. By carefully considering various factors and using the right techniques, it is possible to develop a well-structured plan that helps ensure that the most important requirements are addressed first and the project is completed on time and within budget.

4) Business Process Improvement Techniques.

a) Business process automation (BPA)

Definition: Business process automation (BPA) is the use of technology to automate business processes to improve efficiency and reduce costs. BPA helps to reduce human intervention in business processes and helps save time and costs.

Benefits: The benefits of BPA include improved efficiency, reduced costs, increased accuracy, and reduced errors. BPA also improves tracking and control of business processes and helps businesses make data-driven decisions.

Application to TuneSource: In the Tune Source project, BPA can be applied to automate business processes such as music inventory management, order processing, and customer management.

Here are some examples of applying BPA to the Tune Source project:

- **Automatic music store management**: The system can automatically update the song store when new tracks are added or deleted and alert the staff when the number of songs in the archive reaches a certain threshold.

- **Automated order processing**: The system can automatically process orders from customers, including checking the validity of payment information, confirming orders and sending shipping information to customers. .

- **Automated customer management**: The system can automatically manage customer information, including updating contact information, tracking purchase history and sending promotional information to customers.

b) Business process improvement (BPI)

Business Process Improvement (BPI) is a method used to improve the efficiency and performance of business processes. BPI involves analyzing current processes, identifying problems and opportunities for improvement, and redesigning processes to achieve better results.

Some of the benefits of applying BPI to a TuneSource project may include:

- ✓ **Increase efficiency**: By optimizing business processes, reduce the time and costs associated with performing daily tasks.
- ✓ **Reduce costs**: By eliminating or automating manual steps in business processes, reduce labor costs and increase efficiency.
- ✓ **Improve customer satisfaction**: By improving product and service quality, increase customer satisfaction and attract more new customers.

Here are some examples of applying BPI to a TuneSource project:

- ❖ **Search Engine Optimization**: To help customers easily find the song they are looking for, TuneSource can analyze search patterns and optimize the search function. This may include refining the search algorithm, implementing filtering and sorting options, and providing intelligent search suggestions.
- ❖ **User Experience Improvement:** TuneSource may collect feedback from customers and use that information to improve the user experience. This could include improving the user interface, enhancing customer support, and providing more flexible payment options.
- ❖ **Optimize order processing:** TuneSource can analyze current order processing and find ways to improve it to reduce processing time and increase efficiency. This could include automating some steps in the process, using new technology to track orders, and improving delivery processes.

c) Business process reengineering (BPR)

Business Process Reengineering (BPR) is a method used to improve the performance of business processes by completely redesigning them. BPR differs from other process improvement methods such as Business Process Improvement (BPI) in that it emphasizes the complete reengineering of existing processes rather than merely improving them.

Some of the benefits of applying BPR to a TuneSource project may include:

- **Increased efficiency**: By completely restructuring business processes, TuneSource is able to reduce the time and costs associated with performing daily tasks.
- **Reduce costs**: By eliminating or automating manual steps in business processes, TuneSource can reduce labor costs and increase efficiency.
- **Improve customer satisfaction**: By improving the quality of products and services, TuneSource can increase customer satisfaction and attract more new customers.

Here are some examples of applying BPR to a TuneSource project:

- **Completely refactored search functionality**: Instead of just optimizing the current search functionality, TuneSource can completely refactor this functionality to achieve better results. This may include implementing a new search system that uses new technology to analyze data and provide more accurate search results.
- **Completely Reengineering User Experience**: Instead of just improving the existing user interface, TuneSource can completely refactor the user experience to achieve better results. This may include implementing a new user interface, using new technology to enhance customer support, and providing more flexible payment options.
- **Complete refactoring of order processing**: Instead of just optimizing existing order processing, TuneSource can completely refactor this process to achieve better results. This may include implementing a new order processing system, using new technology to track orders, and improving delivery processes.

VI. Conclude.

In this P5, we learned about the Tune Source project and the importance of requirements definition in software development. We've learned about requirements basics, different types of requirements, and criteria for assessing requirements importance. We also learned about requirements gathering techniques and how to apply them to a Tune Source project.

Accurate and complete requirements definition is an important step in ensuring that software is developed that meets user needs and operates efficiently. By using sound requirements gathering techniques and classifying requirements by importance, we can ensure that the most important requirements are addressed first and the project is completed on time. and within the budget.

Hopefully the information in this P5 will help you better understand the definition of requirements in software development and how to apply it to the Tune Source project.

P6: Use appropriate software analysis tools/techniques to carry out a software investigation and create supporting documentation.

I. Use Case Diagrams.

1) Introduction to Use Case Diagrams.

Use Case Diagram is a type of behavioral diagram that provides a visual representation of the interactions between users (actors) and a system. It shows the different use cases (functions or features) provided by the system and the relationships between actors and use cases.

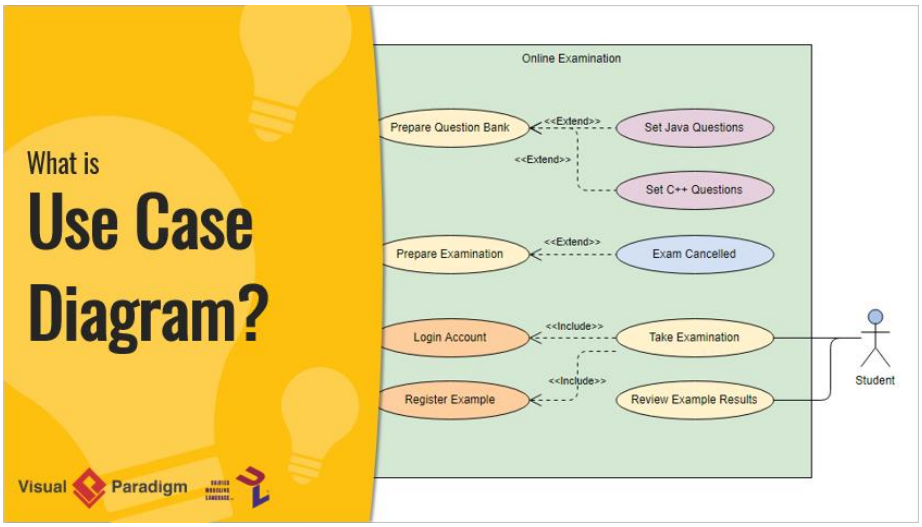


Figure 1: Use case Diagram

The purpose of a Use Case Diagram is to help identify and organize the requirements of a system. By creating a Use Case Diagram, you can gain a better understanding of the different users of the system, their goals, and how they interact with the system to achieve those goals. This can help guide the development of the software, ensuring that it meets the needs of its users.

Use Case Diagrams can be used in software analysis and design to model the behavior of a system. They are often used in the early stages of software development to help identify and prioritize requirements, as well as to communicate these requirements to stakeholders. Use Case Diagrams can also be used throughout the development process to validate that the software is meeting its requirements and to guide testing and maintenance efforts.

2) Components of a Use Case Diagram.

The main components of a Use Case Diagram are actors, use cases, and relationships.

- **Actors**: Actors represent the different types of users that interact with the system. They can be human users, such as customers or administrators, or external systems that interact with the system. In a Use Case Diagram, actors are typically represented by stick figures.

- **Use cases**: Use cases represent the different functions or features provided by the system. They describe the actions that an actor can perform within the system to achieve a specific goal. In a Use Case Diagram, use cases are typically represented by ovals.

- **Relationships**: Relationships represent the connections between actors and use cases, as well as between different use cases. There are several types of relationships that can be used in a Use Case Diagram, including include, extend, and generalization. An include relationship indicates that one use case is included within another use case. An extend relationship indicates that one use case can optionally extend the behavior of another use case. A generalization relationship indicates that one use case is a more general version of another use case.

Here is an example of a Use Case Diagram for an online shopping system:

In this diagram, the Customer actor can perform several use cases, including Search Item, View Item,

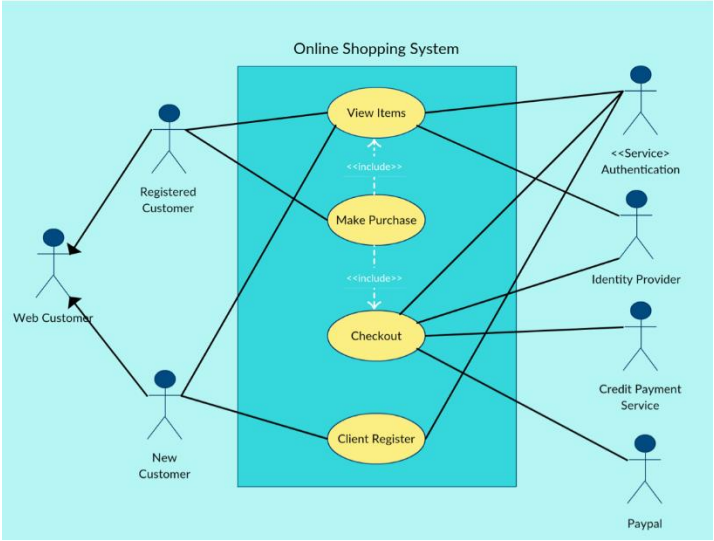


Figure 2: Use case diagram of online shopping system

Add to Cart, and Checkout. The View Item use case includes the Add to Cart use case, indicating that a customer can add an item to their cart while viewing it. The Checkout use case leads to the Pay use case, indicating that a customer can pay for their items after checking out.

3) Creating a Use Case Diagram.

To create a Use Case Diagram, you can follow these steps:

Step 1 - Identify the actors: Start by identifying the different types of users that will interact with the system. These can be human users, such as customers or administrators, or external systems

that interact with the system. Consider the goals and needs of each actor to help guide the development of the use cases.

Step 2 - Identify the use cases: Next, identify the different functions or features that the system will provide to meet the needs of its actors. Each use case should represent a specific action that an actor can perform within the system to achieve a specific goal.

Step 3 - Define relationships: Once you have identified the actors and use cases, define the relationships between them. This can include relationships such as include, extend, and generalization. Consider how each use case relates to other use cases and how they support the goals of the actors.

Step 4 - Organize the diagram: Finally, organize the diagram in a clear and logical manner. Place actors on the outside of the diagram, with use cases in the center. Use lines to connect actors to their associated use cases and to show relationships between use cases. Use consistent notation and labeling to make the diagram easy to understand.

4) Best Practices for Use Case Diagrams.

Here are some best practices for creating effective Use Case Diagrams:

- **Use consistent notation**: Use a standard notation, such as the Unified Modeling Language (UML), to create your Use Case Diagrams. This will make it easier for others to understand your diagrams and help ensure that they are interpreted correctly.
- **Keep diagrams simple and easy to understand**: Avoid cluttering your diagrams with too many details or complex relationships. Instead, focus on the most important actors, use cases, and relationships, and organize the diagram in a clear and logical manner.
- **Validate diagrams with stakeholders**: Share your Use Case Diagrams with stakeholders, such as users, developers, and managers, to ensure that they accurately represent the requirements of the system. Use their feedback to refine and improve your diagrams.
- **Iterate and refine**: Creating effective Use Case Diagrams is an iterative process. Don't be afraid to make changes and refinements as you learn more about the system and its requirements.

By following these best practices, you can create Use Case Diagrams that are clear, accurate, and effective in communicating the requirements of a system.

5) Applying Use Case Diagrams to a Project of TuneSource.

a) Customers

Identify the actors: In the Tune Source project, one of the main actors is the Customer.

Identify the use cases: The use cases for this system might include Search Music, Listen to Music Samples, Purchase Individual Downloads, Establish Customer Subscription Account, and Purchase Music Download Gift Cards.

Define relationships: Once you have identified the actor and use cases, define the relationships between them. For the Tune Source project, this might include relationships such as an include relationship between Search Music and Listen to Music Samples (indicating that a customer can listen to music samples while searching for music), and a sequence of relationships between Purchase Individual Downloads, Establish Customer Subscription Account, and Purchase Music Download Gift Cards (indicating that a customer can purchase individual downloads or establish a subscription account before purchasing music download gift cards).

Organize the diagram: Finally, organize the diagram in a clear and logical manner. Place the actor on the outside of the diagram, with use cases in the center. Use lines to connect the actor to their associated use cases and to show relationships between use cases. Use consistent notation and labeling to make the diagram easy to understand.

Here is an example of how these steps might be applied to create a Use Case Diagram for the Tune Source project:

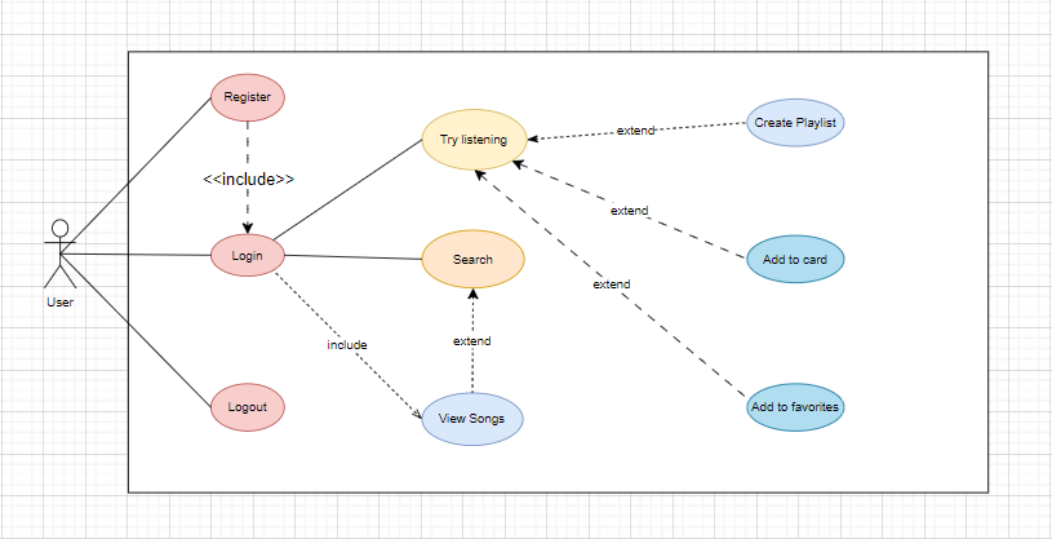


Figure 3: Use case diagram of user

b) Administrators

Identify the actors: In the Tune Source project, one of the main actors is the Admin.

Identify the use cases: The use cases for this system might include Add Music, Edit Music, Delete Music, Manage Artists, Manage Music Genres, and Manage Albums.

Define relationships: Once you have identified the actor and use cases, define the relationships between them. For the Tune Source project, this might include relationships such as a sequence of relationships between Add Music, Edit Music, and Delete Music (indicating that an admin can add, edit, or delete music), and a sequence of relationships between Manage Artists, Manage Music Genres, and Manage Albums (indicating that an admin can manage various aspects of Tune Source's music catalog).

Organize the diagram: Finally, organize the diagram in a clear and logical manner. Place the actor on the outside of the diagram, with use cases in the center. Use lines to connect the actor to their associated use cases and to show relationships between use cases. Use consistent notation and labeling to make the diagram easy to understand.

Here is an example of how these steps might be applied to create a Use Case Diagram for the Tune Source project:

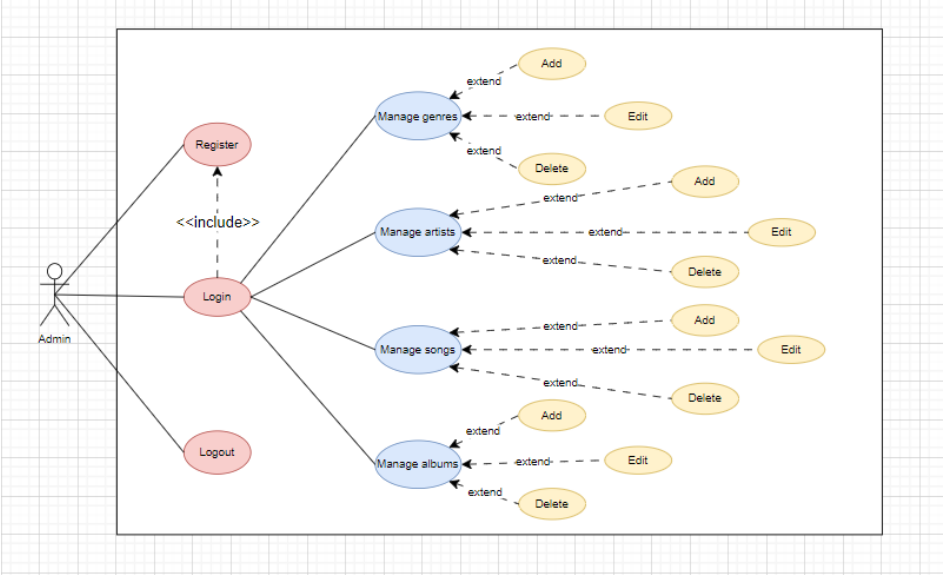


Figure 4: Use case diagram of admin

II. Data flow Diagram.

1) Introduction to Data Flow Diagrams.

Data Flow Diagrams (DFDs) are a graphical tool used to describe the flow of data in a system. They show how data is entered, processed, stored, and output by the system. DFDs can help identify bottlenecks or inefficiencies in data flows and can be used to design more efficient systems.

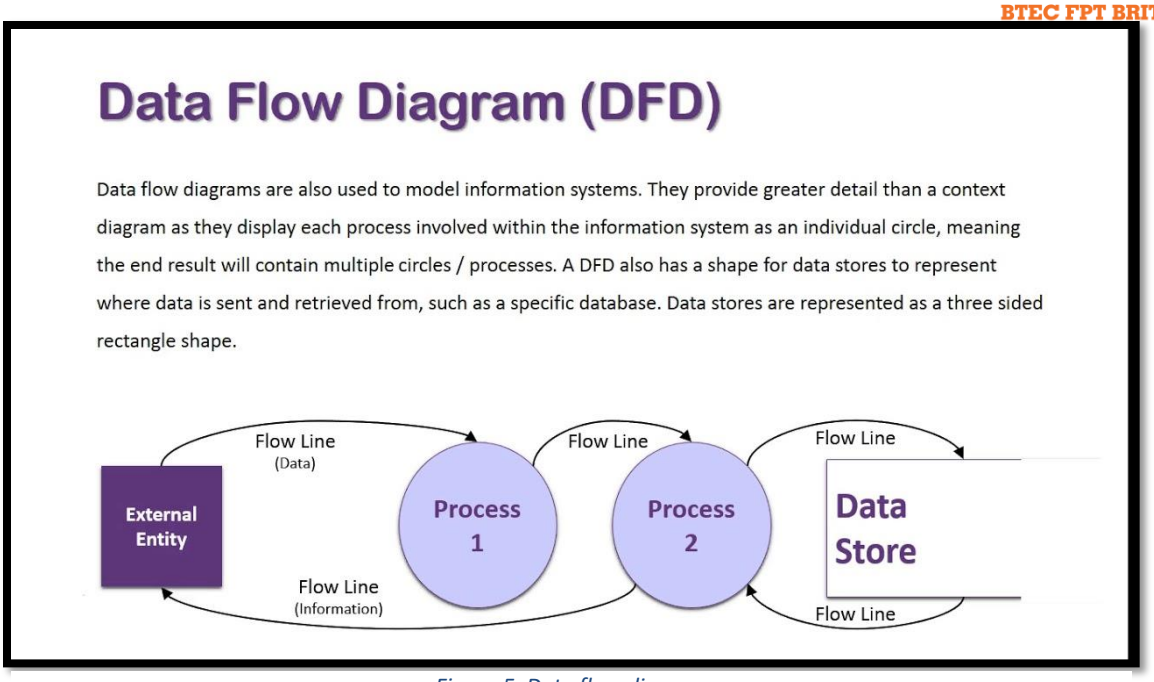


Figure 5: Data flow diagram

The purpose of DFDs is to provide an overview of how data moves through the system and how it is processed. They help software analysts and designers better understand the system's requirements and design its functionality.

DFDs can be used in software analysis and design to determine system requirements, design data structures, and build data processing algorithms. They can also be used to communicate with various stakeholders, including customers, users and regulators, to ensure that everyone has a clear understanding of how the system works.

2) Components of a Data Flow Diagram.

A Data Flow Diagram (DFD) consists of many key components, including processes, data flows, data warehouses, and external entities.

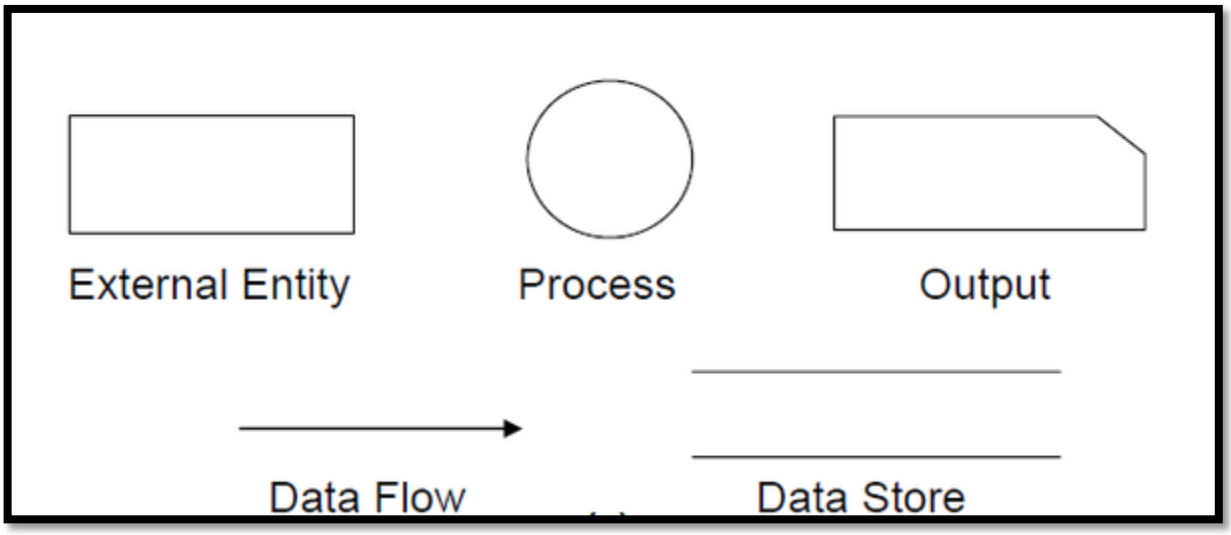


Figure 6: Components of a data flow diagram

Processes: Processes represent the operations or transformations that occur in the system, such as computation or data manipulation. The process is represented by a circle or rectangle with rounded edges.

Data Flow: Data Flow represents the movement of data between processes, data warehouses, and external entities. Data flows are represented by arrows, with the arrowhead being the source and the arrowhead being the destination.

Data Warehouse: A data warehouse represents places where data is stored in a system, such as a database or files. The datastore is represented by a rectangle with two short sides missing or in other words lying within two parallel lines.

External Entities: External entities that represent the source or destination of data outside the system, such as users or other systems. External entities are represented by rectangles with their respective names.

Example: Suppose we have a simple system that allows users to enter personal information and store them in a database. In this case, we have an "Input" process, a data stream from "User" to an "Information Input" process, a data stream from an "Information Entry" process to "Information Engine" process. database", and a datastore "Database". A DFD for this system might look like the following image:

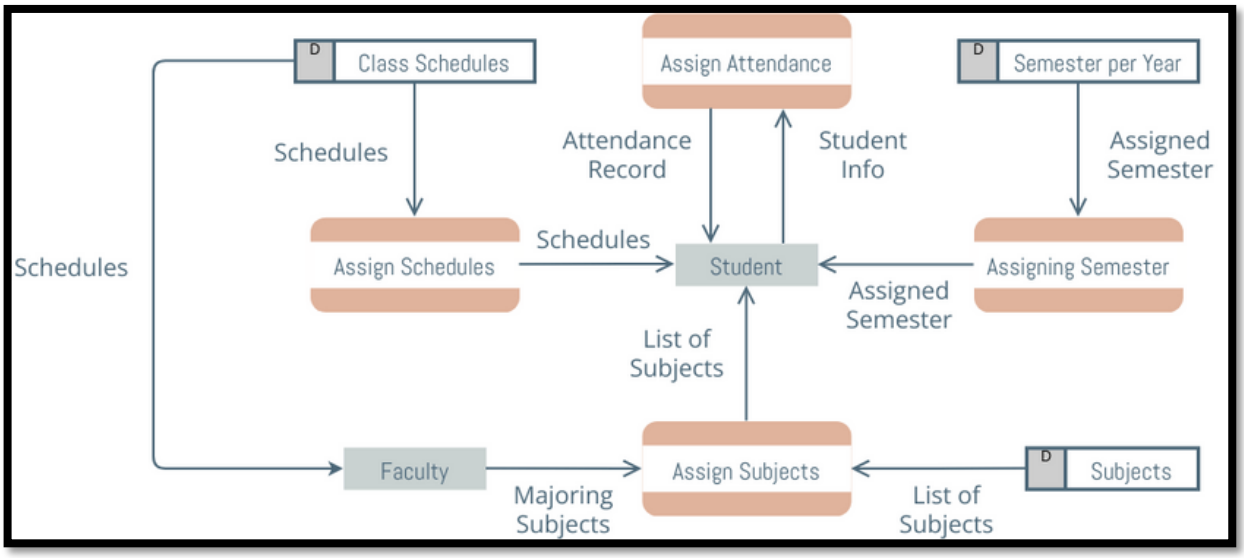


Figure 7: Data flow diagram of student management

3) Creating a Data Flow Diagram.

To create a Data Flow Diagram (DFD), you need to perform the following steps:

- Identify processes, data flows, data stores, and external entities: The first step in creating a DFD is to identify the key components of the system, including processes, data flows, and repositories. data and external entities. This can be done by analyzing the system requirements and identifying the activities, data and users involved.
- Identify the relationships between components: After you have identified the main components of the system, you need to define the relationships between them. This includes defining how data moves between processes, data stores, and external entities, as well as how processes interact with each other.
- Organize your diagrams clearly and logically: Finally, you need to organize your diagrams in a clear and logical way. This means arranging components so that they are easy to see and understand, and using consistent symbols and symbols to represent different components.

Once you have completed the above steps, you should have a complete DFD for your system. This DFD can be used to better understand how the system works and to design its functionality.

4) Best Practices for Data Flow Diagrams.

Here are some best practices for creating effective Data Flow Diagrams (DFDs):

- **Use consistent notation**: Using consistent notation and symbols when creating a DFD is important because it helps to make the diagram easy to understand. This means using standard symbols for processes, data flows, data stores, and external entities, as well as using consistent labeling and naming conventions. For example, processes can be

represented by circles or rounded rectangles, data flows by arrows, data stores by open-ended rectangles, and external entities by rectangles. By using these standard symbols consistently throughout the diagram, readers can quickly understand the meaning of each element.

- **Keep diagrams simple and easy to understand**: DFDs should be kept as simple as possible while still accurately representing the system. This means avoiding unnecessary detail and complexity, and organizing the diagram in a clear and logical manner. For example, it may be helpful to group related processes together or to use color-coding to highlight important elements. By keeping the diagram simple and well-organized, readers can easily understand the flow of data within the system.
- **Validate diagrams with stakeholders**: Validating DFDs with stakeholders is important because it helps to ensure that the diagram accurately represents the system. This can be done by reviewing the diagram with stakeholders such as customers, users, and business analysts, and soliciting feedback and suggestions for improvement. By incorporating this feedback into the diagram, you can create a more accurate and useful representation of the system.

By following these best practices, you can create effective Data Flow Diagrams that accurately represent the system and are easy to understand. These diagrams can be a valuable tool for understanding and designing complex systems.

5) Applying Data Flow Diagrams to a project of TuneSource.

To apply Data Flow Diagrams (DFDs) to a specific project, such as the Tune Source project, you can follow these steps:

- Identify processes, data flows, data stores, and external entities: The first step in creating a DFD for the Tune Source project is to identify the main components of the system, including processes, data flows, data stores, and external entities. For example, some of the processes in the Tune Source system might include "Search for Music," "Listen to Music Samples," and "Purchase Downloads." Data flows might include the flow of search queries from the user to the system, the flow of music samples from the system to the user, and the flow of payment information from the user to the system. Data stores might include a database of music tracks and a database of customer information. External entities might include customers and payment processors.
- Create a Context Diagram: Once you have identified the main components of the system, you can create a Context Diagram that shows the overall data flow within the system. This diagram should include all of the external entities that interact with the system, as well as a single process that represents the entire system. Data flows between the external entities and the system process should be shown.

- Create Level 0 and Level 1 diagrams: After creating a Context Diagram, you can create more detailed Level 0 and Level 1 diagrams that show how data flows between specific processes within the system. A Level 0 diagram shows all of the main processes within the system and how they are connected by data flows. A Level 1 diagram provides even more detail by breaking down each process into its sub-processes and showing how data flows between them.

By following these steps, I can create a set of DFDs that provide a clear and detailed representation of how data flows within the Tune Source system as follow:

a) Context Diagram.

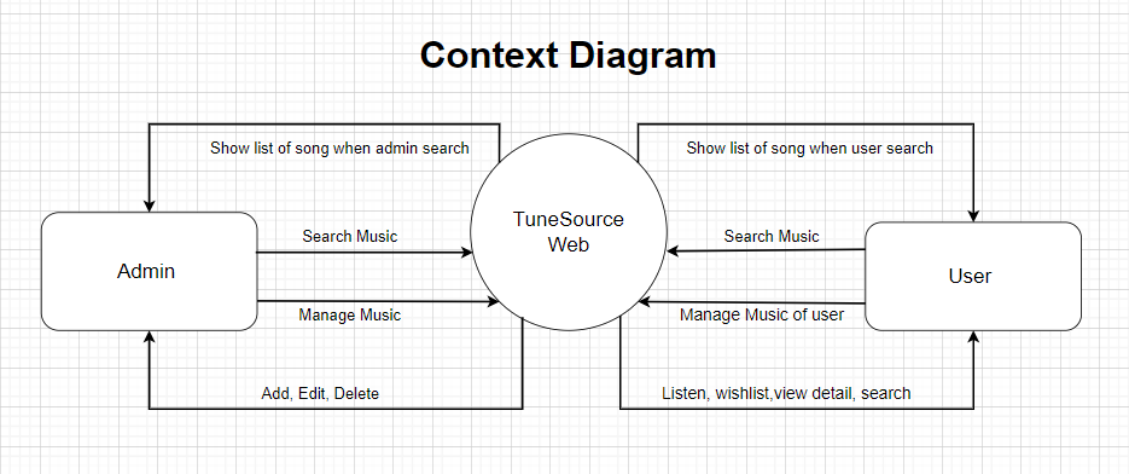


Figure 8: Context Diagram

b) Level 0

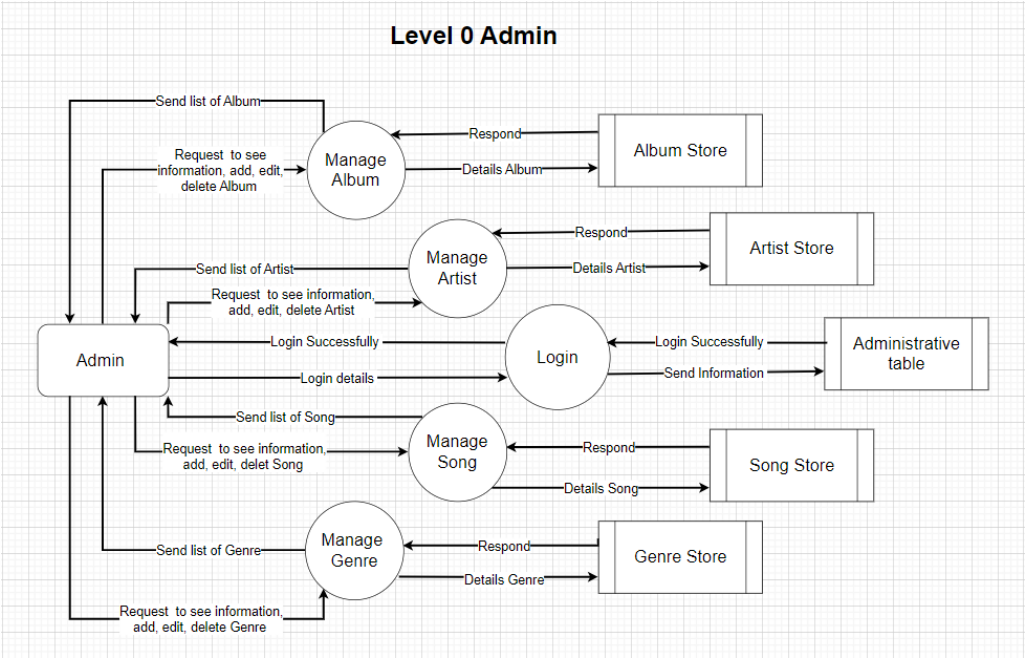


Figure 9: Level 0 Admin

c) Level 1

III. Entity Relation Diagram.

1) Introduction to Entity-Relationship Diagrams.

An Entity-Relationship Diagram (ERD) is a type of diagram that shows how entities, such as people, objects, or concepts, relate to each other within a system or a database. Entities are the different data components that are labeled and connected by symbols. ERDs are used to design, debug, or understand relational databases and other applications that involve large sets of data.

ER diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research. They use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships and their attributes. They mirror grammatical structure, with entities as nouns and relationships as verbs.

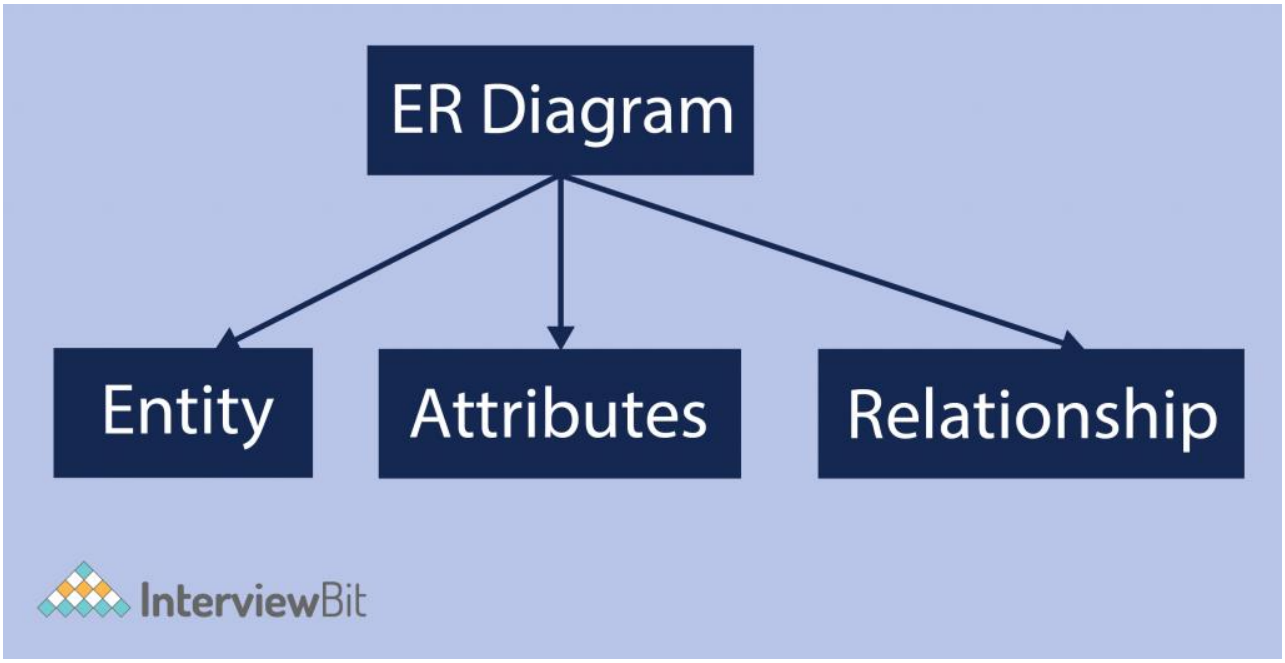


Figure 10: Entity Relation Diagram

In summary, ERDs provide a visual representation of the relationships between entities in a system or database. They are an important tool for software analysis and design, helping developers to understand the structure of the data they are working with and to design effective and efficient databases.

2) Components of an Entity-Relationship Diagram.

An ERD is composed of three main elements: entities, attributes, and relationships.

- **Entities**: Entities are the building blocks of an ERD and represent objects, persons, concepts, or events that contain data. They are typically displayed in a rectangle and can be thought of as the "nouns" in a database. For example, in a university database, entities might include students, courses, and instructors.
- **Attributes**: Attributes are the characteristics or properties of an entity. They are displayed in a circle or an oval and provide more detail about the entity. For example, the attributes of a student entity might include their name, student ID number, and major.
- **Relationships**: Relationships define how entities are related to each other. They are represented by lines connecting entities and can be thought of as the "verbs" in a database. For example, a relationship between a student entity and a course entity might be "enrolled in" or "teaches".
- **Primary Key**: A primary key is a unique identifier for each record in an entity. It is represented with an underline or a unique symbol in an ERD. Primary keys ensure that each entity instance can be uniquely identified.

- **Foreign Key**: A foreign key is an attribute in one entity that refers to the primary key of another entity, establishing a link between the two entities. It is often represented with an asterisk (*) or a symbol in the ERD.
- **Cardinality**: Cardinality specifies the number of instances of one entity that can be associated with another entity. For example, in a "one-to-many" relationship between "Author" and "Book," an author

In addition to these main elements, there are also additional elements based on these main elements, such as weak entities, multi-valued attributes, derived attributes, weak relationships, and recursive relationships. These elements provide more detail and flexibility when modeling complex systems.

Overall, ERDs provide a visual representation of the relationships between entities in a system or database. By understanding the components of an ERD and how they work together, you can design effective and efficient databases.

3) Creating an Entity-Relationship Diagram.

Creating an ERD involves several steps, including identifying entities, attributes, and relationships, defining the cardinality and optionality of relationships, and organizing the diagram in a clear and logical manner.

- **Identifying entities**: The first step in creating an ERD is to identify the entities that will be included in the diagram. Entities are typically objects, persons, concepts, or events that contain data. For example, in a university database, entities might include students, courses, and instructors.
- **Identifying attributes**: Once you have identified the entities, the next step is to identify their attributes. Attributes are the characteristics or properties of an entity and provide more detail about the entity. For example, the attributes of a student entity might include their name, student ID number, and major.
- **Identifying relationships**: After identifying the entities and their attributes, the next step is to identify the relationships between entities. Relationships define how entities are related to each other and can be thought of as the "verbs" in a database. For example, a relationship between a student entity and a course entity might be "enrolled in" or "teaches".
- **Defining cardinality and optionality**: Once you have identified the relationships between entities, you need to define their cardinality and optionality. Cardinality refers to the number of instances of one entity that can be associated with instances of another entity. Optionality refers to whether or not a relationship is required or optional.

- **Organizing the diagram**: Finally, once you have identified all of the components of your ERD, you need to organize them in a clear and logical manner. This involves arranging the entities, attributes, and relationships in a way that makes sense and is easy to understand.

Overall, creating an ERD involves identifying the components of your system or database and organizing them in a clear and logical manner. By following these steps, you can create an effective and efficient ERD that accurately represents your data.

4) Best Practices for Entity-Relationship Diagrams.

Some best practices for creating effective Entity-Relationship Diagrams (ERDs):

- **Use consistent notation**: When creating an ERD, it's important to use a consistent notation throughout the diagram. This means using the same symbols and conventions to represent entities, attributes, and relationships. This makes it easier for others to understand your diagram and reduces the risk of confusion or misinterpretation.
- **Keep diagrams simple and easy to understand**: ERDs can quickly become complex and difficult to understand if they include too many entities, attributes, and relationships. To avoid this, try to keep your diagrams as simple as possible while still accurately representing your data. This might involve breaking a large diagram into several smaller diagrams or using color-coding or other visual cues to help organize the information.
- **Validate diagrams with stakeholders**: Before finalizing your ERD, it's a good idea to validate it with stakeholders such as developers, database administrators, or business analysts. This can help you identify any errors or omissions in your diagram and ensure that it accurately represents the data and relationships in your system.
- **Use software tools**: There are many software tools available that can help you create and manage ERDs. These tools can help you create diagrams more quickly and easily than drawing them by hand, and they often include features such as automatic layout and formatting, version control, and collaboration tools.

Overall, by following these best practices, you can create effective and efficient ERDs that accurately represent your data and are easy for others to understand.

5) Applying Entity-Relationship Diagrams to a Project.

To apply Entity-Relationship Diagrams to the Tune Source project, you can follow these steps:

- **Identify the entities**: Start by identifying the different entities (objects or concepts) that are relevant to the system. For the Tune Source project, these might include entities such as Customer(User), Song, Artist, Genre, and Album.
- **Identify the attributes**: Next, identify the attributes (properties or characteristics) of each entity. For example, the Customer entity might have attributes such as Name, Email, and

Subscription Status, while the Music entity might have attributes such as Title, Artist, and Genre.

- **Identify the relationships**: Once you have identified the entities and their attributes, identify the relationships between them. For example, there might be a relationship between the Customer and Music entities to represent that a customer can purchase individual music downloads or subscribe to a monthly service for unlimited downloads.
- **Define the cardinality and optionality of relationships**: For each relationship, define its cardinality (the number of instances of one entity that can be associated with an instance of another entity) and optionality (whether an instance of one entity must be associated with an instance of another entity). For example, the relationship between the Customer and Music entities might have a cardinality of many-to-many (a customer can purchase many music downloads and a music download can be purchased by many customers) and be optional (a customer does not have to purchase any music downloads).
- **Organize the diagram**: Finally, organize the diagram in a clear and logical manner. Place entities in boxes with their attributes listed inside. Use lines to connect entities that have a relationship and add labels to indicate the cardinality and optionality of each relationship. Use consistent notation and labeling to make the diagram easy to understand.

Here is an example of how these steps might be applied to create an Entity-Relationship Diagram for the Tune Source project:

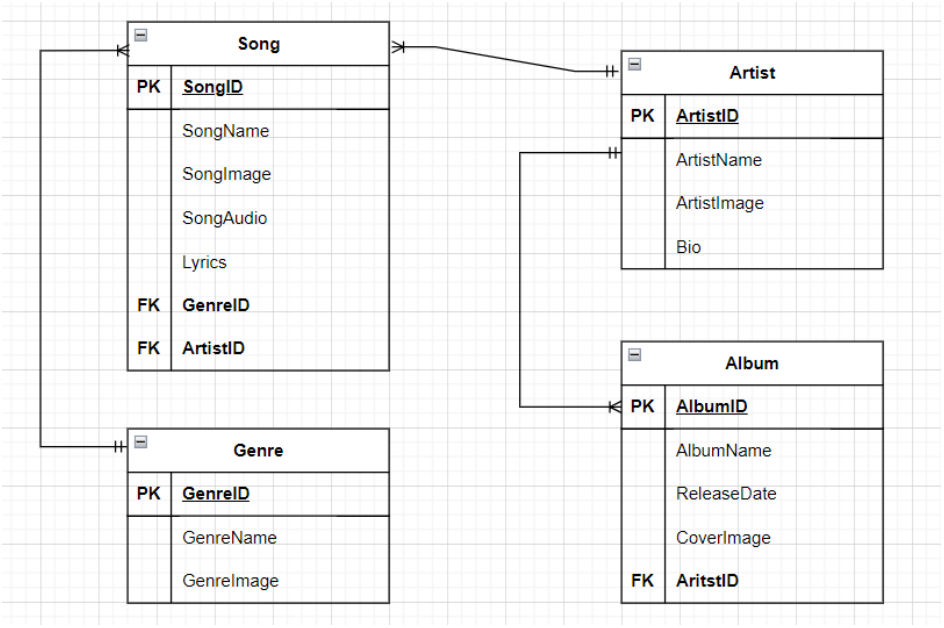


Figure 11: ERD

In this diagram, there are several entities, including Customer(User), Song, Artist, Genre, and Album. Each entity has several attributes listed inside its box. There are relationships between some of the entities, such as between Customer and Music (to represent that a customer can purchase individual music downloads or subscribe to a monthly service for unlimited downloads), and between Music and Artist (to represent that a music download is associated with an artist). The cardinality and optionality of each relationship is indicated by labels on the lines connecting the entities.

P7: Explain how user and software requirements have been addressed.

I. Wireframe of the project.

A wireframe is a low-fidelity visual representation of a software application's user interface. It shows the layout and structure of the different screens and pages, as well as the placement of buttons, menus, and other interactive elements. Wireframes are typically created early in the design process to help define the overall structure and functionality of an application before moving on to more detailed design work.

1. Wireframe for login and registration

a) Login

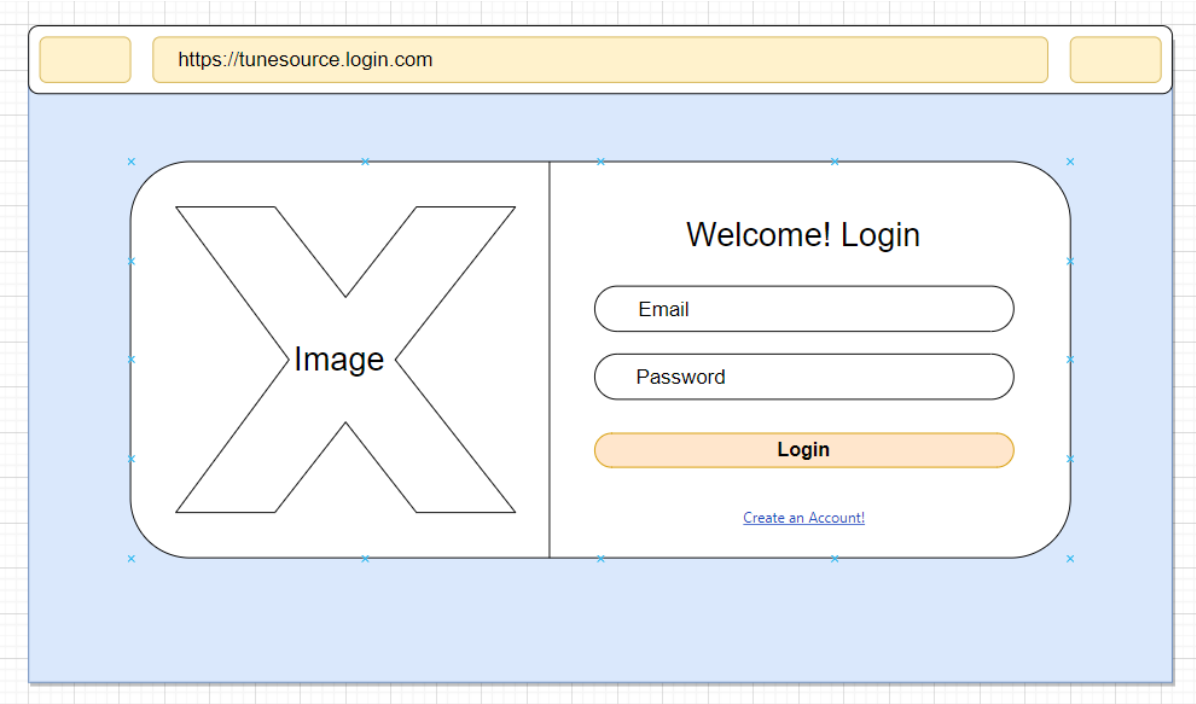


Figure 12: Wireframe login

b) Registration

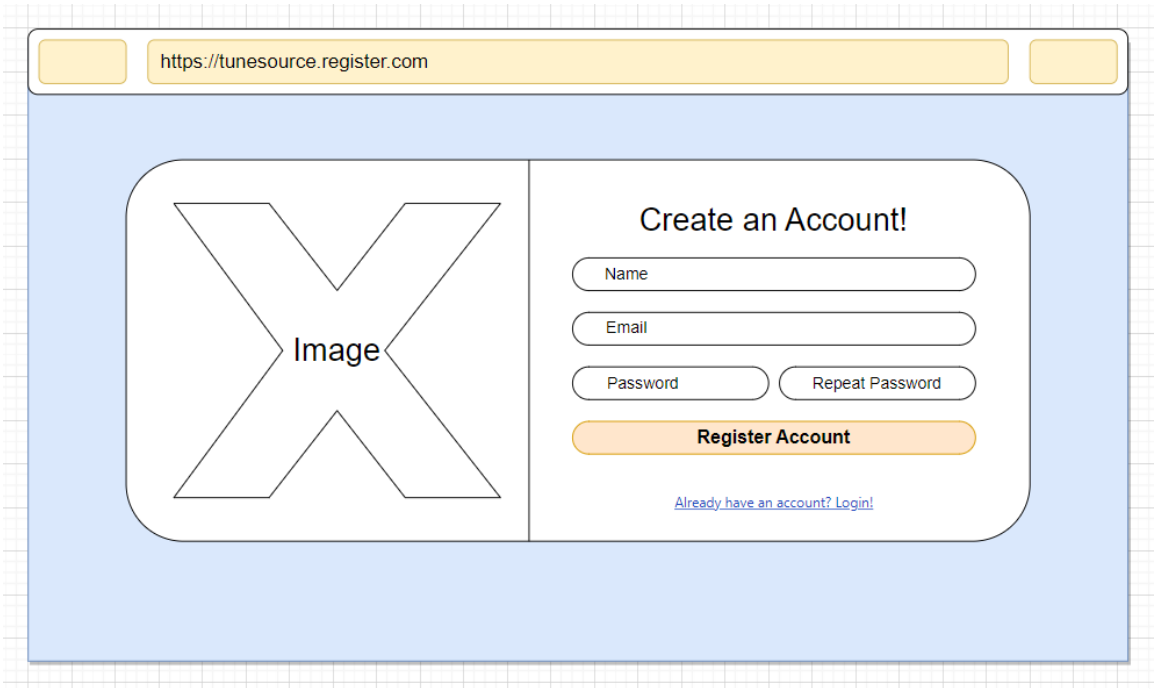


Figure 13: Wireframe register

2. Wireframe for admin page

a) Dashboard

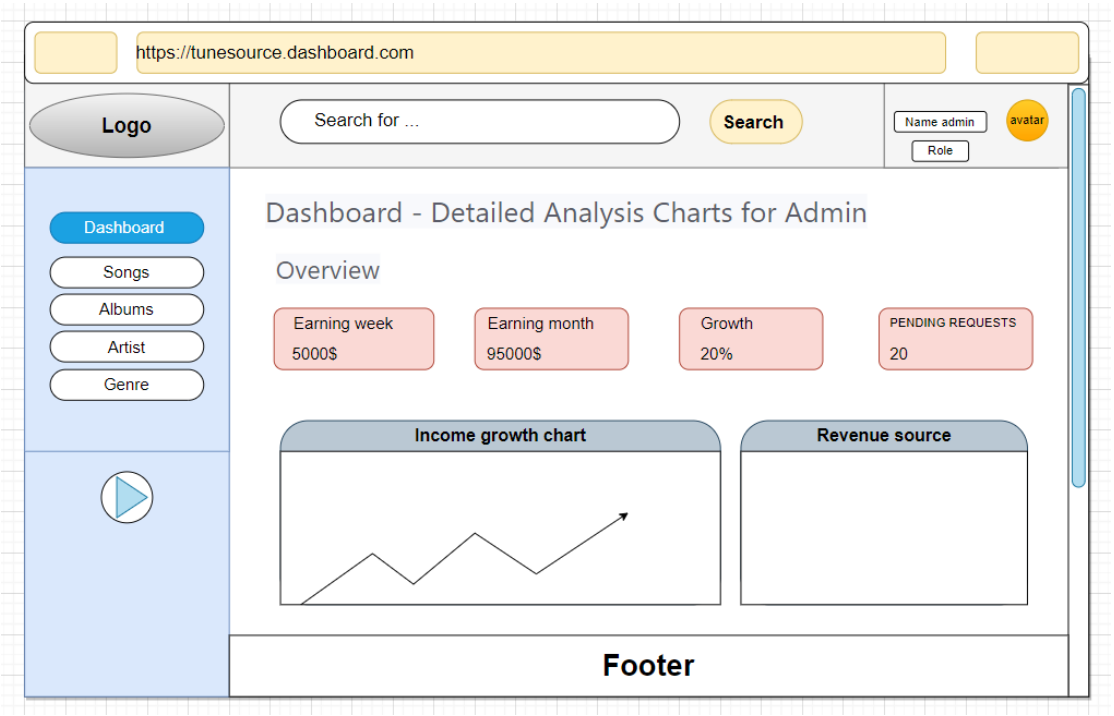


Figure 14: Wireframe dasdboard

b) Album

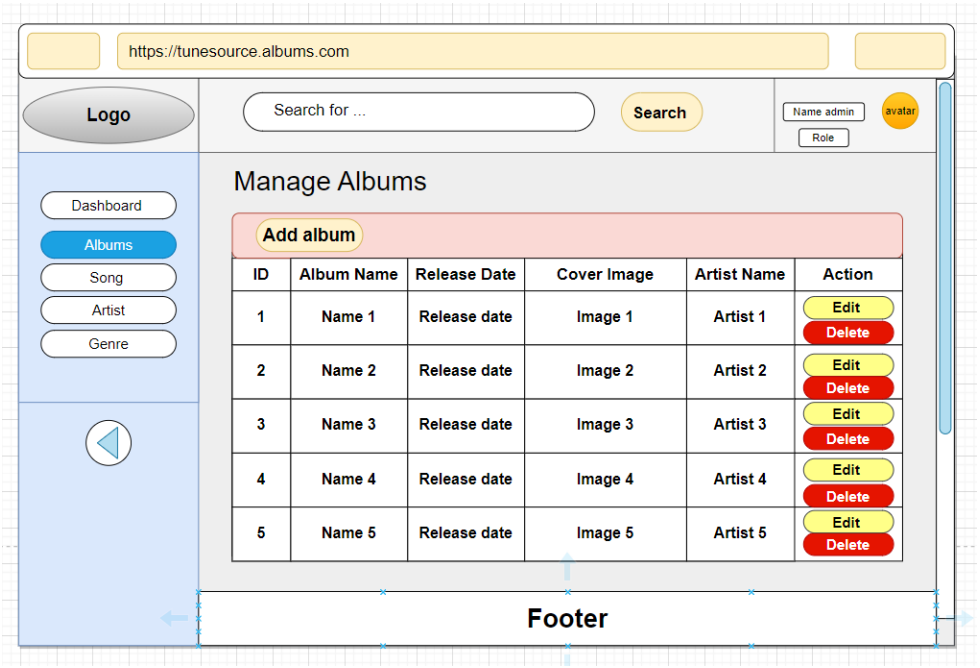


Figure 15: Wireframe album

c) Song

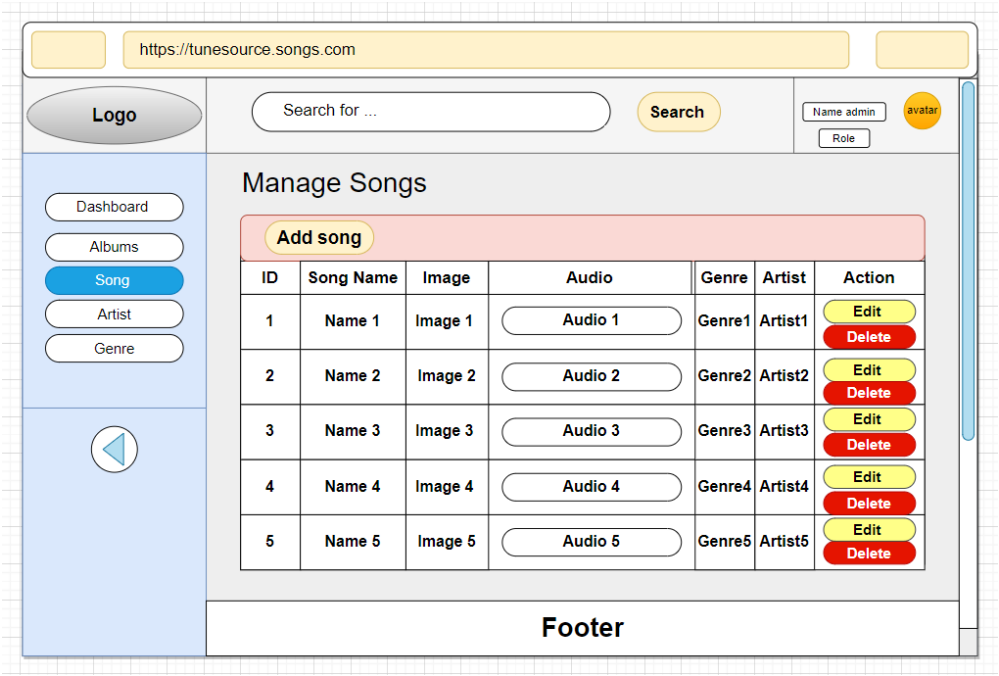


Figure 16: Wireframe song

d) Artist

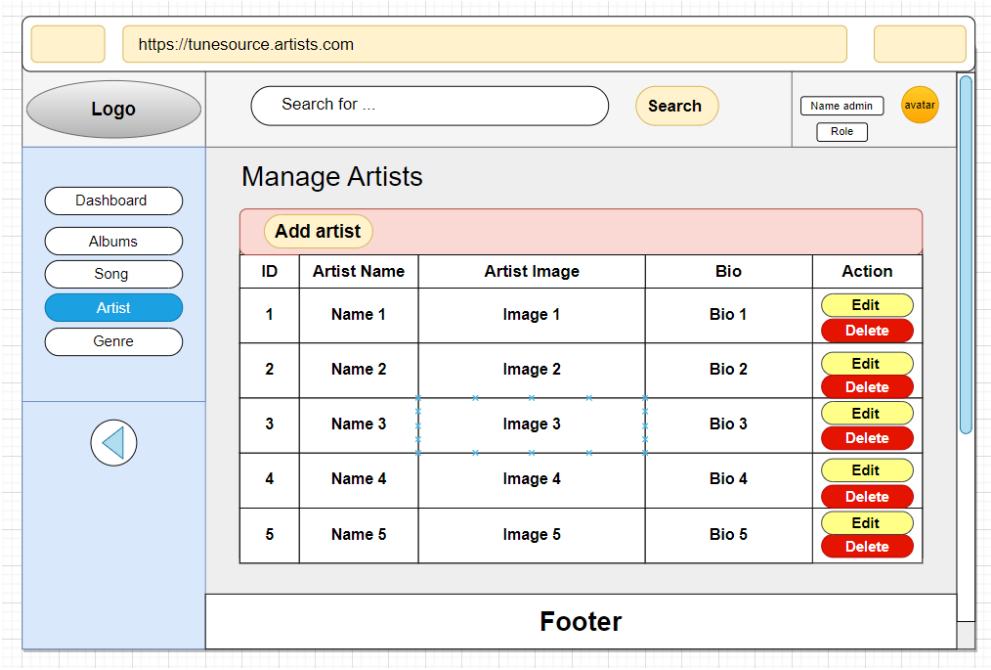


Figure 18: Wireframe artist

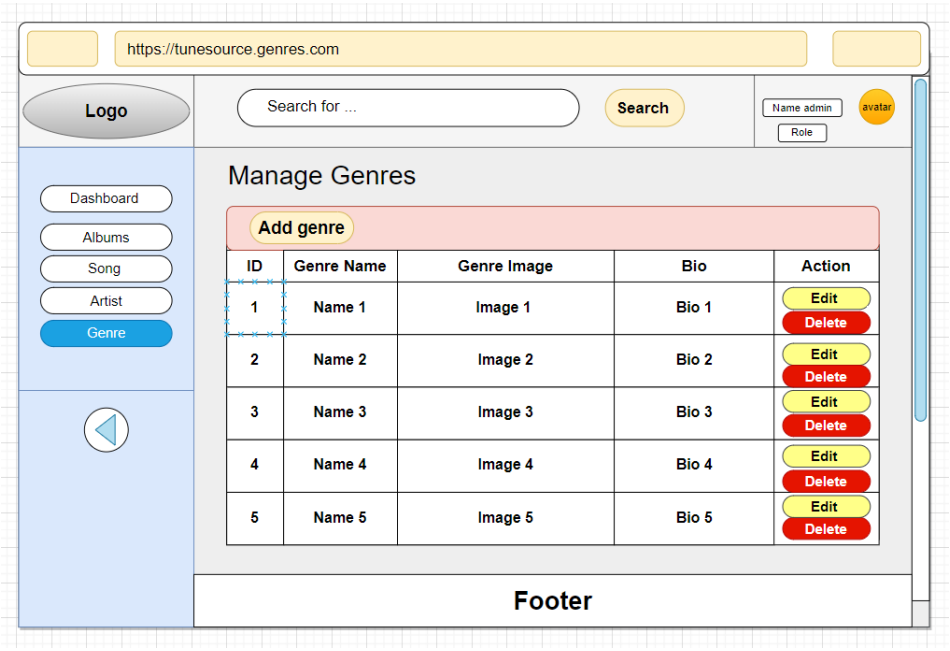


Figure 17: Wireframe genre

e) Genres

II. Database design.

1) Data tables.

a) Admin table.

Field	Type	Description
ID	Int	Primary key
Name	Varchar(255)	Username
Email	Varchar(255)	Email
Password	Varchar(255)	Passwword

Table 1: Table Admin

b) Artist table.

Field	Type	Description
ArtistID	Int	Primary key
ArtistImage	Varchar(255)	Image of artist
ArtistName	Varchar(255)	Name of artist
Bio	Text	Artist Biography

Table 2" Table artist

c) Genre table.

Field	Type	Description

Field	Type	Description
GenreID	Int	Primarykey
GenreName	Varchar(255)	Name of Genre

Table 3: Table genre

d) Album table.

Field	Type	Description
AlbumID	INTEGER	Primarykey
AlbumName	VARCHAR(255)	Name of album
ReleaseDate	DATE	Release date
CoverImage	VARCHAR(255)	Album cover photo
ArtistID	INTEGER	Foreign key refer to table Artist

Table 4: Table album

e) Song table.

Field	Type	Description
SongID	INTEGER	Primarykey
SongImage	Varchar(255)	Song cover photo
SongName	VARCHAR(255)	Name of song

Field	Type	Description
SongAudio	File	Audi of song
Lyrics	TEXT	Lyric of song
GenreID	INTEGER	Foreign key refer to table Genre
ArtistID	INTEGER	Foreign key refer to table Artist

Table 5: Table song

2) Relationships between tables.

There are several relationships between the tables:

- An **Album** belongs to an **Artist** (via the **ArtistID** foreign key in the **Albums** table).
- A **Song** belongs to an **Artist** (via the **ArtistID** foreign key in the **Songs** table) and a **genre** (via the **GenreID** foreign key in the **Songs** table).

3) Architectural design.

Centralization: The main advantage of the client-server model is the built-in centralization capabilities. With this model, all the necessary information is located in a single location. This is very useful for network administrators. Because they have full control and control. With this feature, any problem in the network can be solved in a single place. And so it becomes easier to update resources and data as well.

Security: In the ClientC-Server network, data is well protected thanks to the centralized architecture of the network. It may have access controls in place so that only authorized users can access it. One of the ways to do so is to impose credentials such as Username or Password. Furthermore, if data is lost, files can be restored easily from just a single backup.

Scalability: The client server network model has good scalability. Whenever users need them, they can increase the number of resources, such as the number of clients or servers. As a result, the size of the Server can be easily increased without much interruption.

Accessibility: No distinction between different locations or platforms, every customer can log into the system. This way, all employees can access their company's information without having to use terminal or processor mode.

III. User manual for the project.

1) Instructions for registration and login

a) Register

To register an account on TuneSoure, you need to follow these steps:

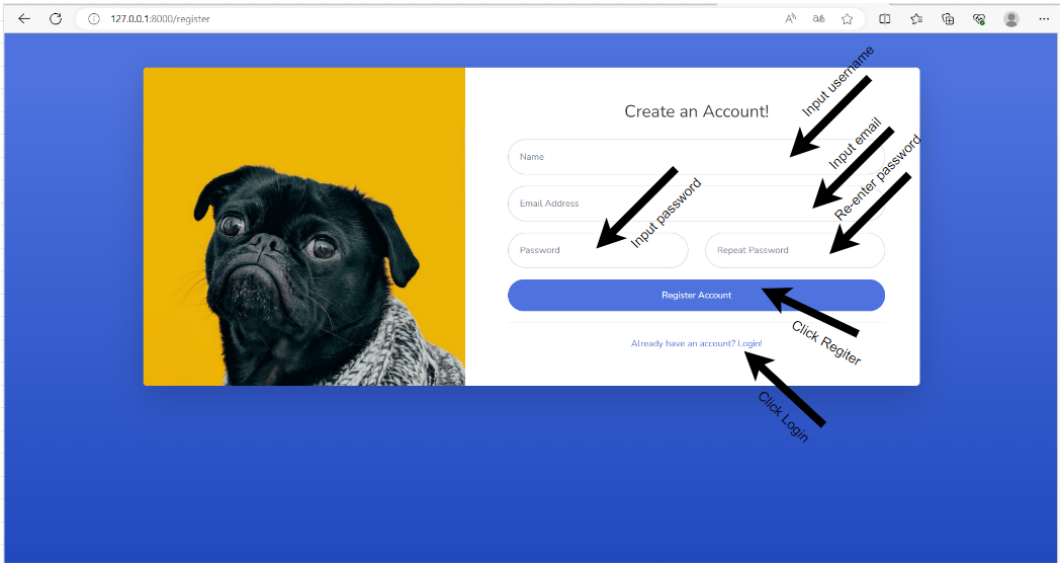


Figure 19: Web register

- Access the website and click the Register button in the upper right corner of the screen.
- Enter your name (display name), email address, password in the corresponding boxes.
- Click the Register button to complete the registration process. You need to fill in the correct information and the correct email form is example@email.com to activate your account.
- After activating your account, you will be redirected to the login page to perform login to the website.

b) Login

To sign in to TuneSoure, you need to follow these steps:

- Enter your email address and password in the respective boxes. Please use the account you registered earlier.
- You need to fill in the fields completely and correctly.
- Click the Sign In button to access your account. If you do not have an account, you can click the register button below, or if you have forgotten your password, you can click the Forgot Password link to retrieve your password via email.
- After logging in, you will be directed to the homepage of TuneSoure, where you can perform the functions of the website.

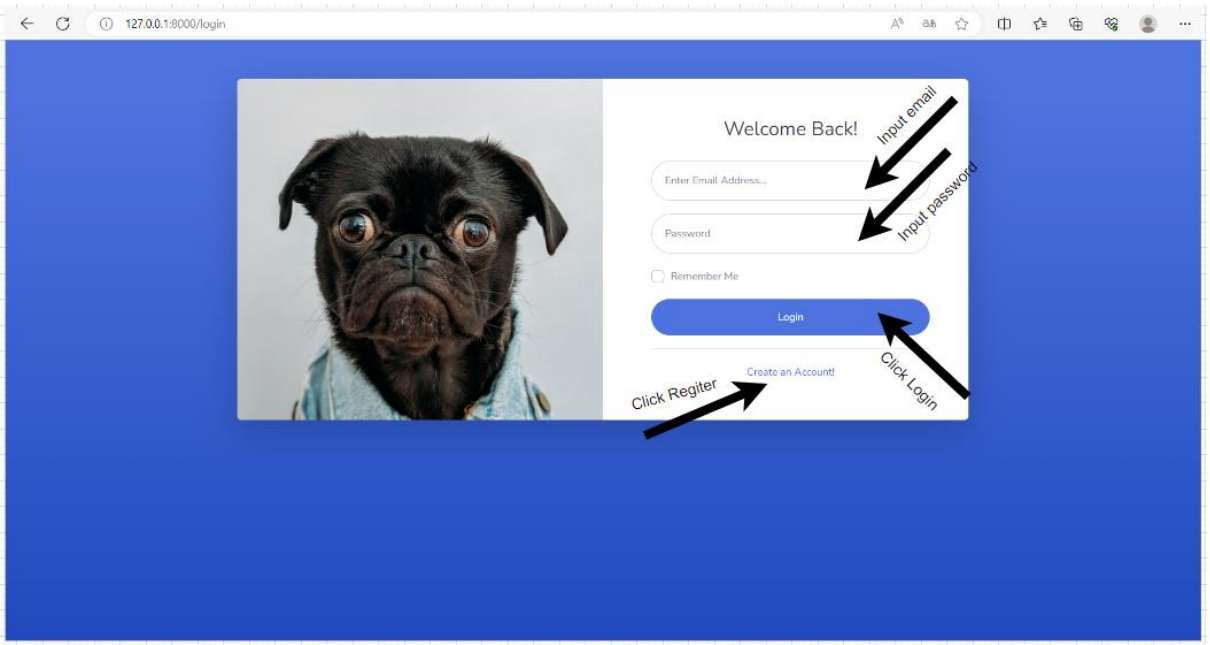


Figure 20: Web login

2) Instructions for adding, editing, and deleting artists

a) Add

After successful login, the admin will be redirected to the dashboard page to see the detailed statistics and revenue of the website, the right side will have a menu bar, click on each item to continue the website's functions:

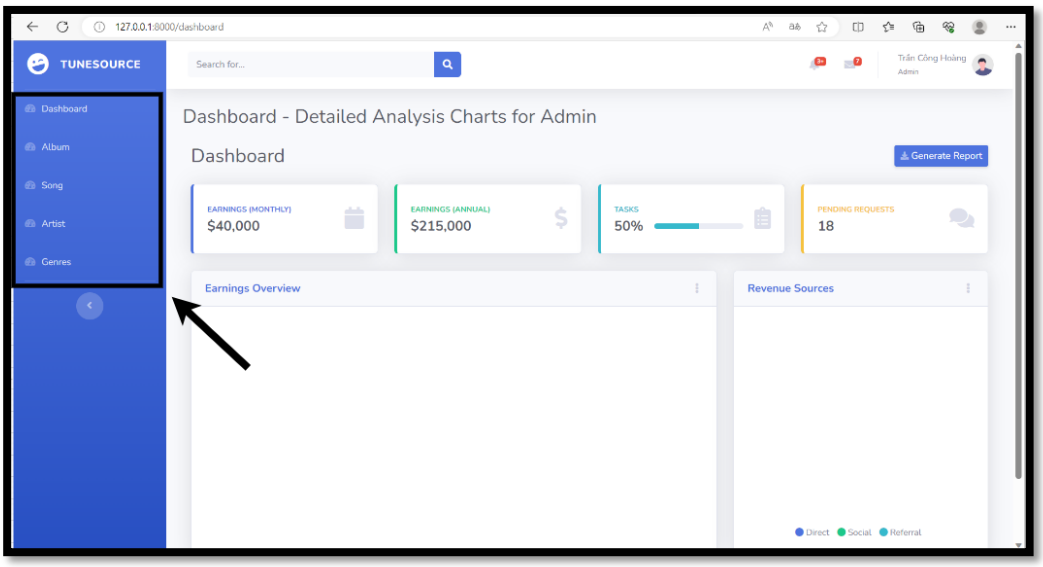


Figure 21: Web dasdhoard

- From the TuneSoure homepage, click the Artist icon in the left navigation bar of the screen.

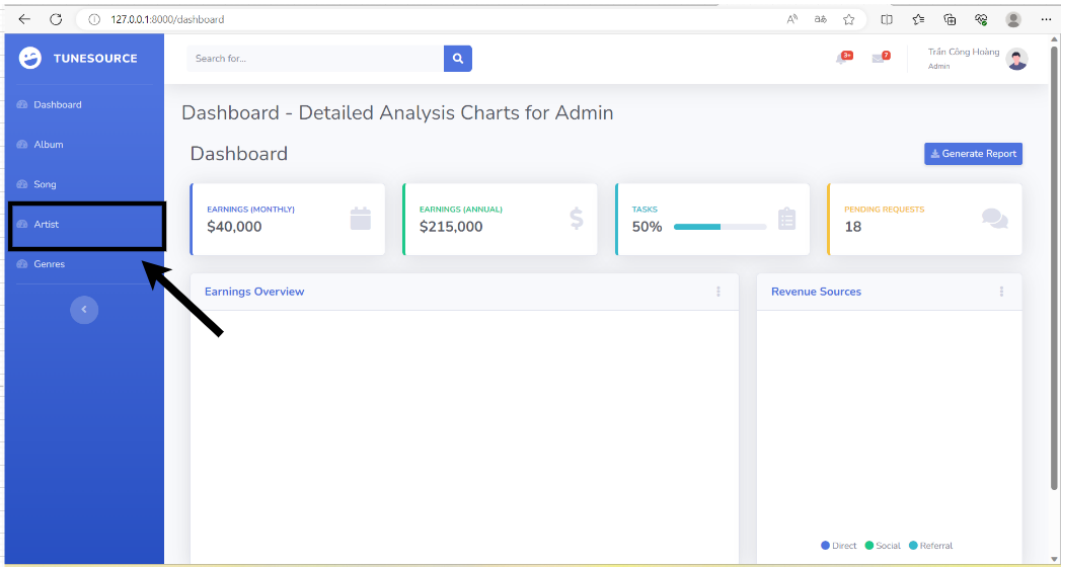


Figure 22: Web artist

- In Artist, to add new Artist, admin will click on Add Artist button.

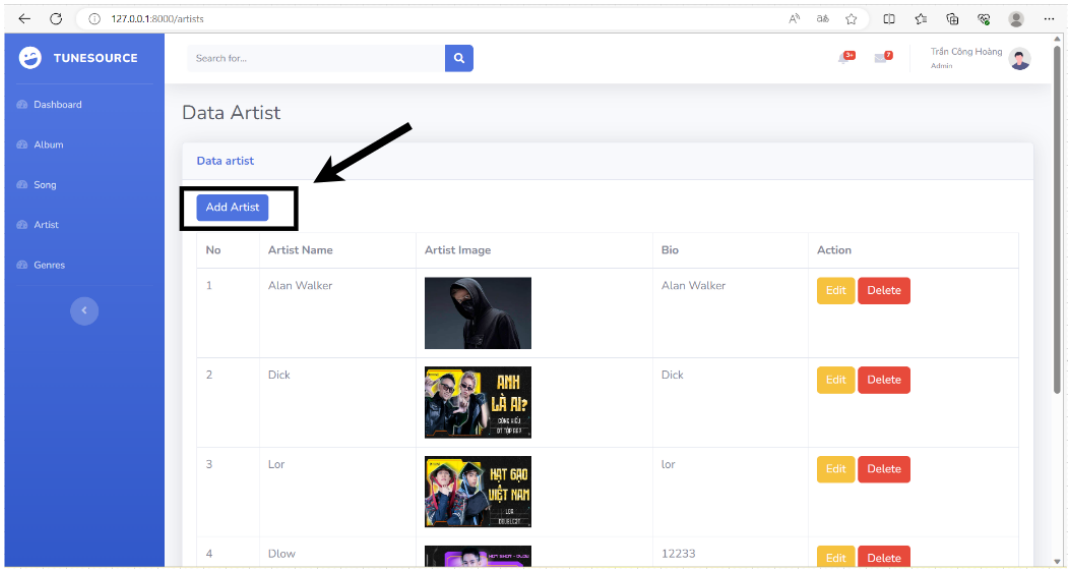


Figure 23: Button Add artist

- After clicking the Add Artist button, the admin will be redirected to the Add New Artist page.

- On the new page, the admin will enter all the required information to add a new Artist.

- In the Artist Image field, the admin will enter by clicking the Chose File button to add an image file, this field requires an image file and has the format of png,jpg,jpeg.

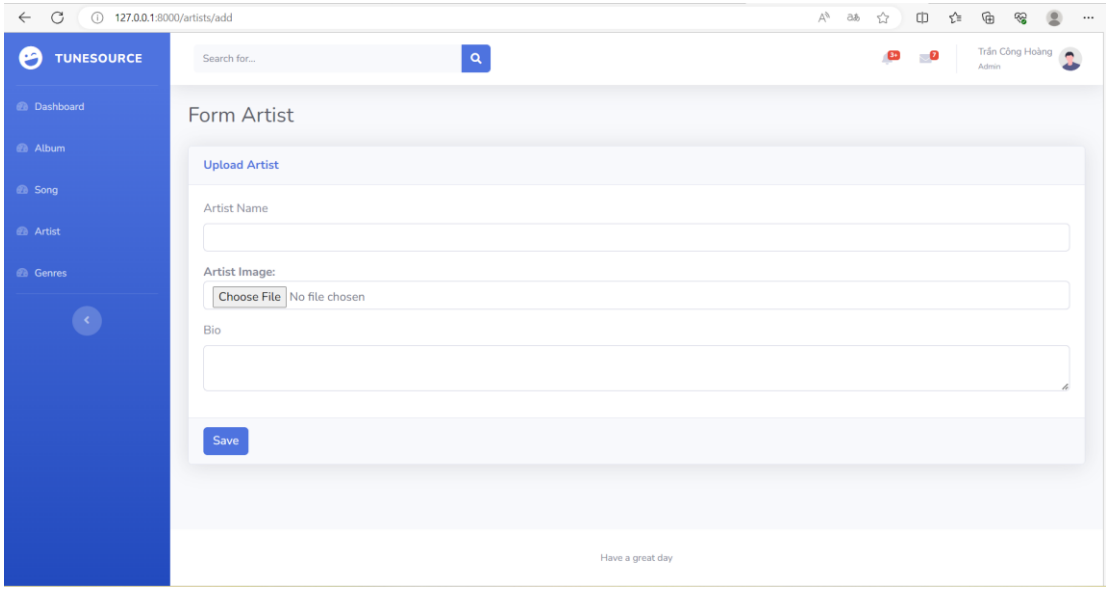


Figure 24: Add artist

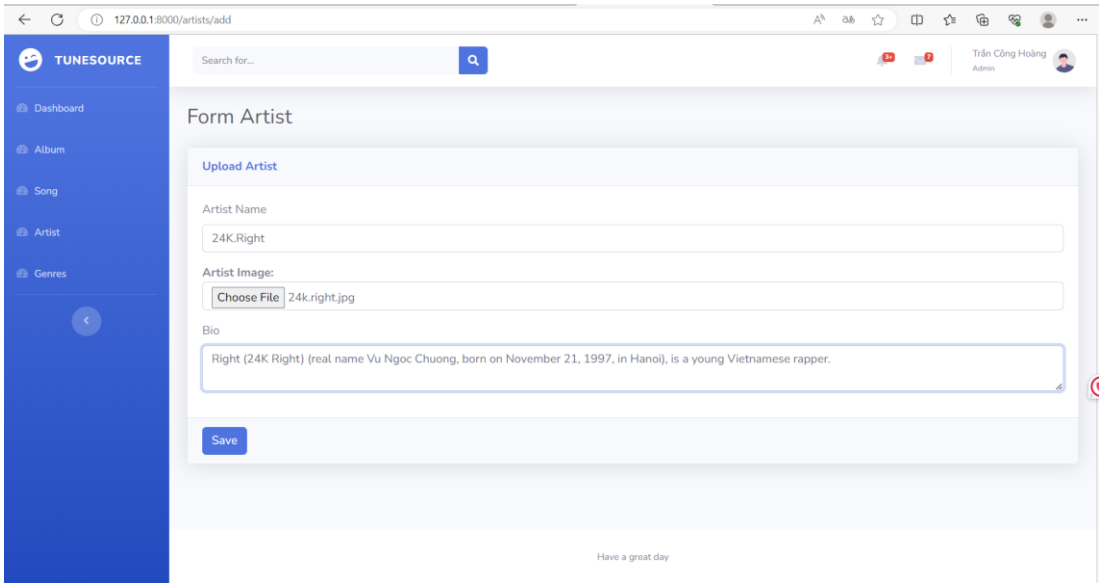


Figure 25: Action add new arrtist

- After entering all the information, press the Save button to save the data and will be redirected to the Artist page, the newly added data will be displayed here.

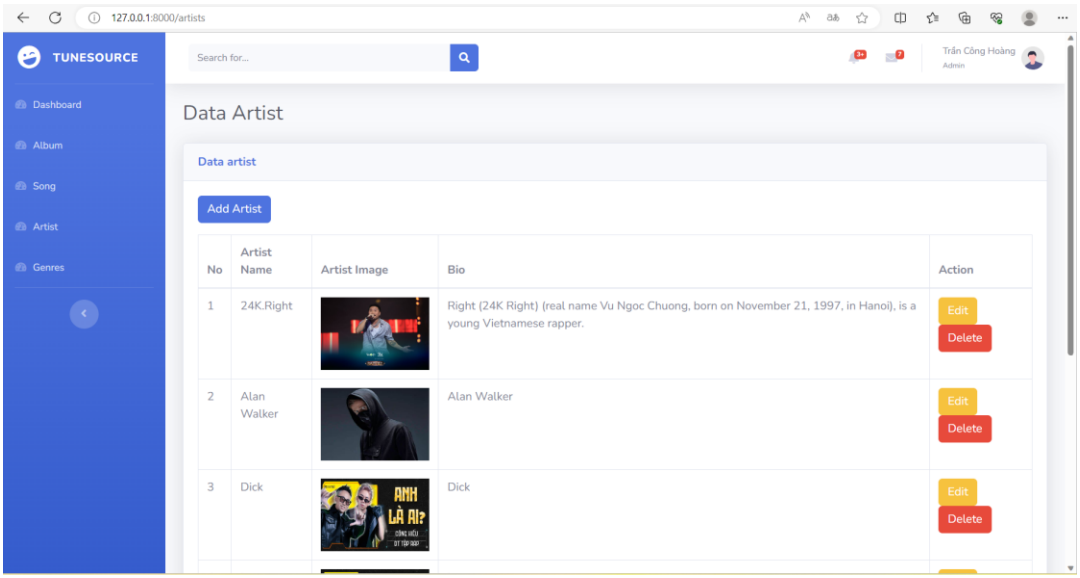


Figure 26: After when add artist success

b) Edit

To perform the function of editing information, the user clicks the Edit button on the right side of the editable information field.

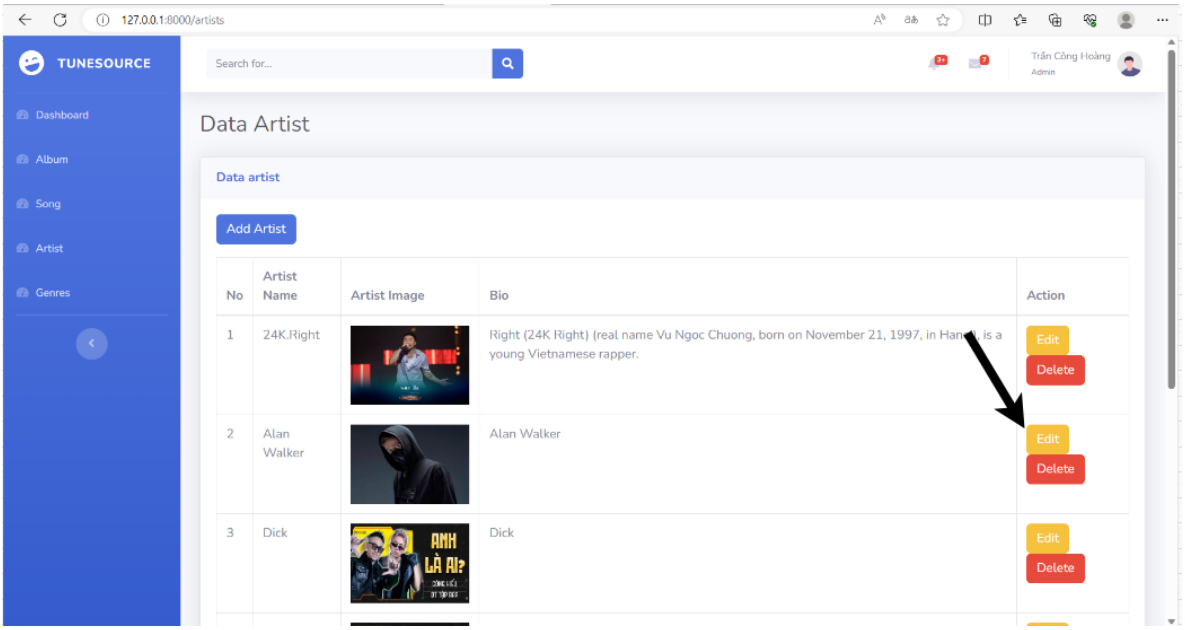


Figure 27: Edit artist

After clicking the Edit button, the user is navigated to the Edit page with the ID being the ID of the selected field.

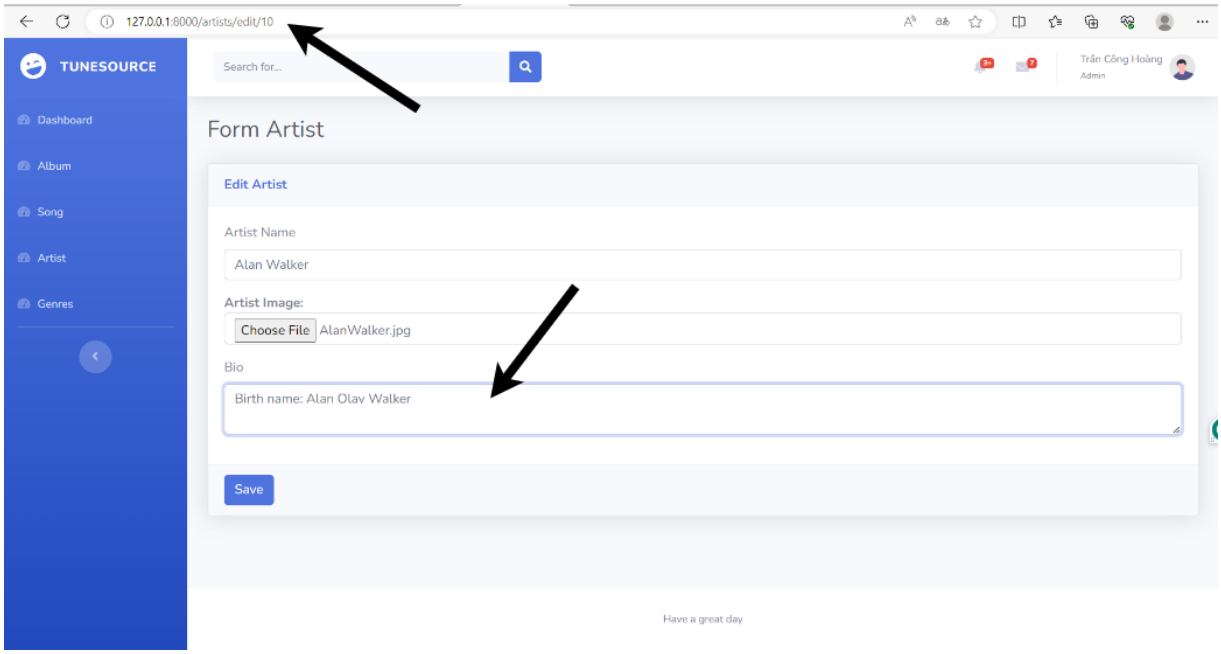


Figure 28: Adit Artist

Here, the user will make changes to the information that needs to be edited

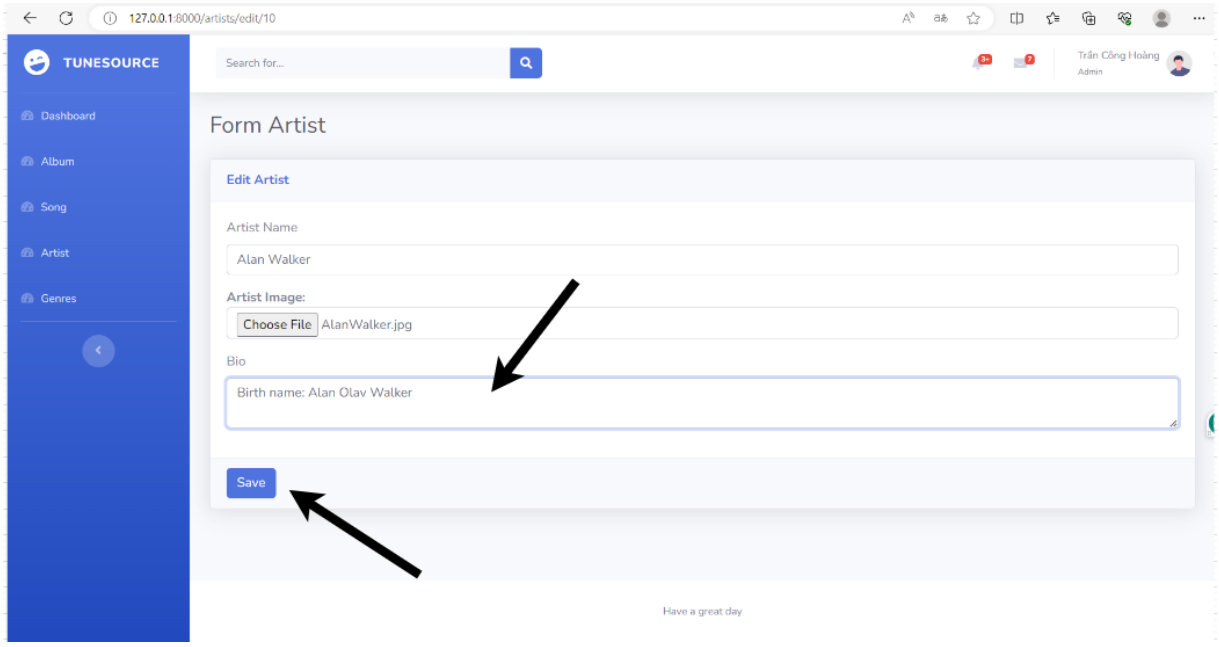


Figure 29: Input information need edit

After successfully changing the information, the user presses the Save button to save the edited data:

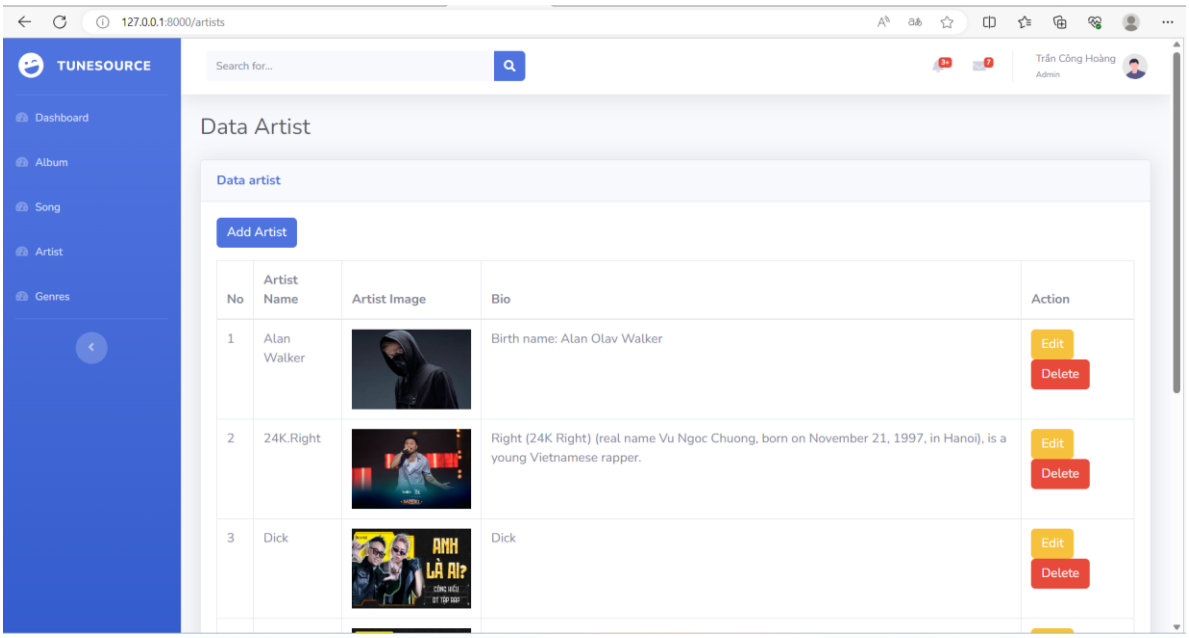


Figure 30: After edit artist

c) Delete

If the save is successful, the user will return to the original page and the edited information will be displayed here.

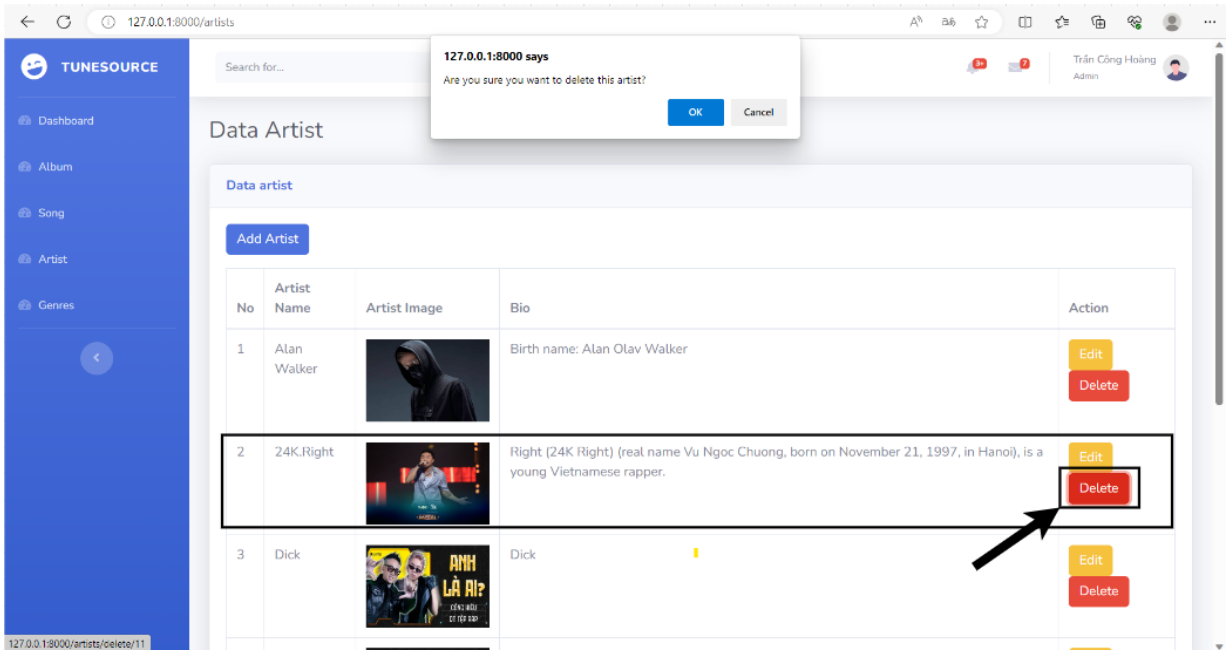


Figure 31: Delete Artist

To perform the delete function, the user clicks the Delete button to the right of the row of information to be deleted.

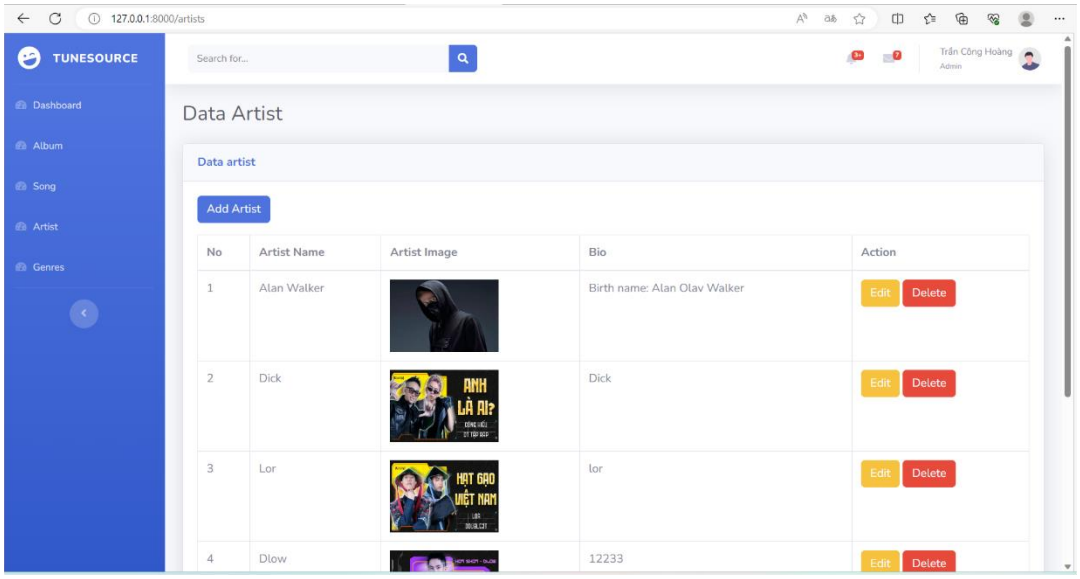


Figure 32: Delete Success

3) Instructions for adding, editing, and deleting genres

a) Add

- To perform Genre information management, click on the Genre icon in the left navigation bar of the screen.

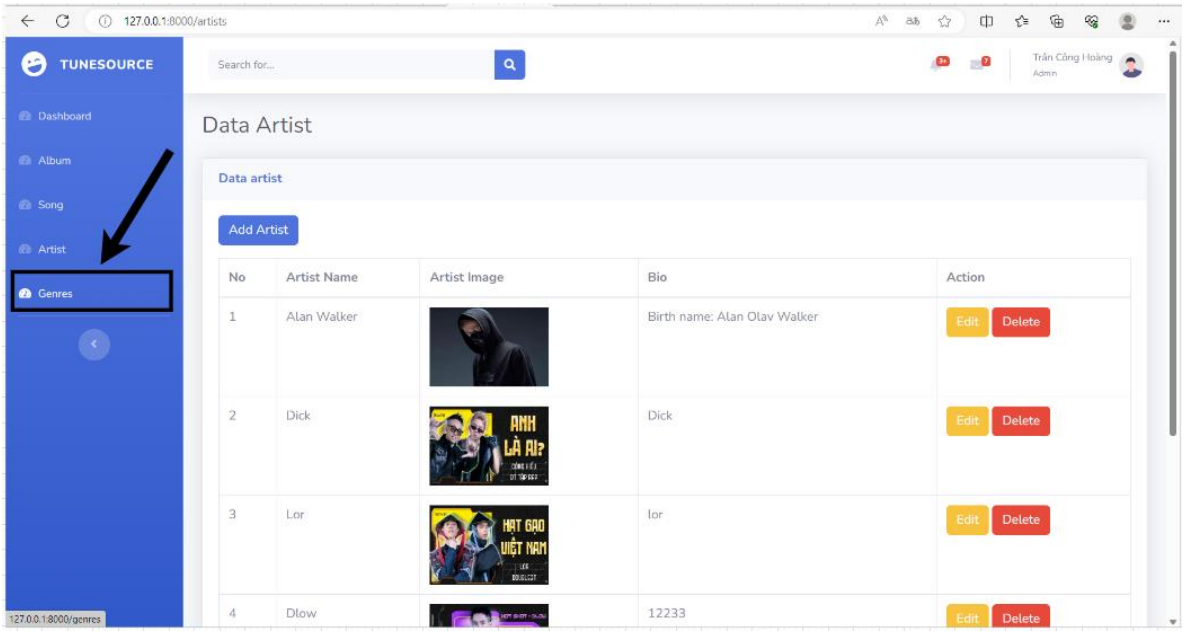


Figure 33: Add genre

- In Genre, to add a new Genre, admin will click on the Add Genre button.

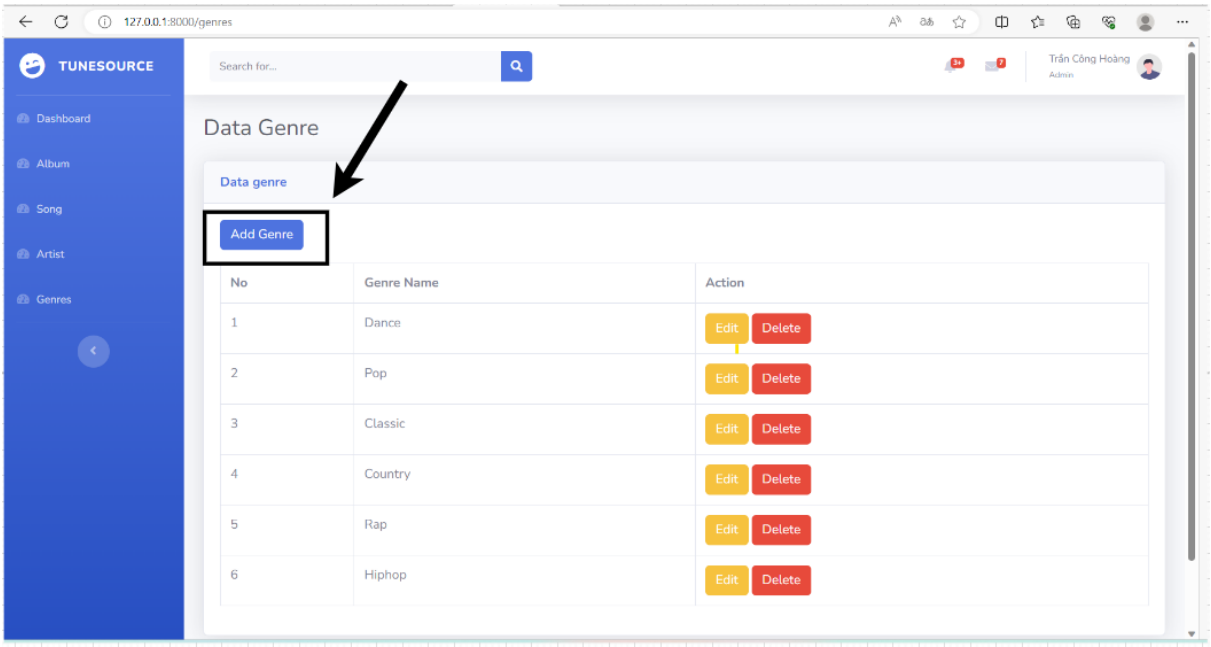


Figure 34: Click add genre

- After clicking the Add Genre button, the admin will be redirected to the new Genre page.

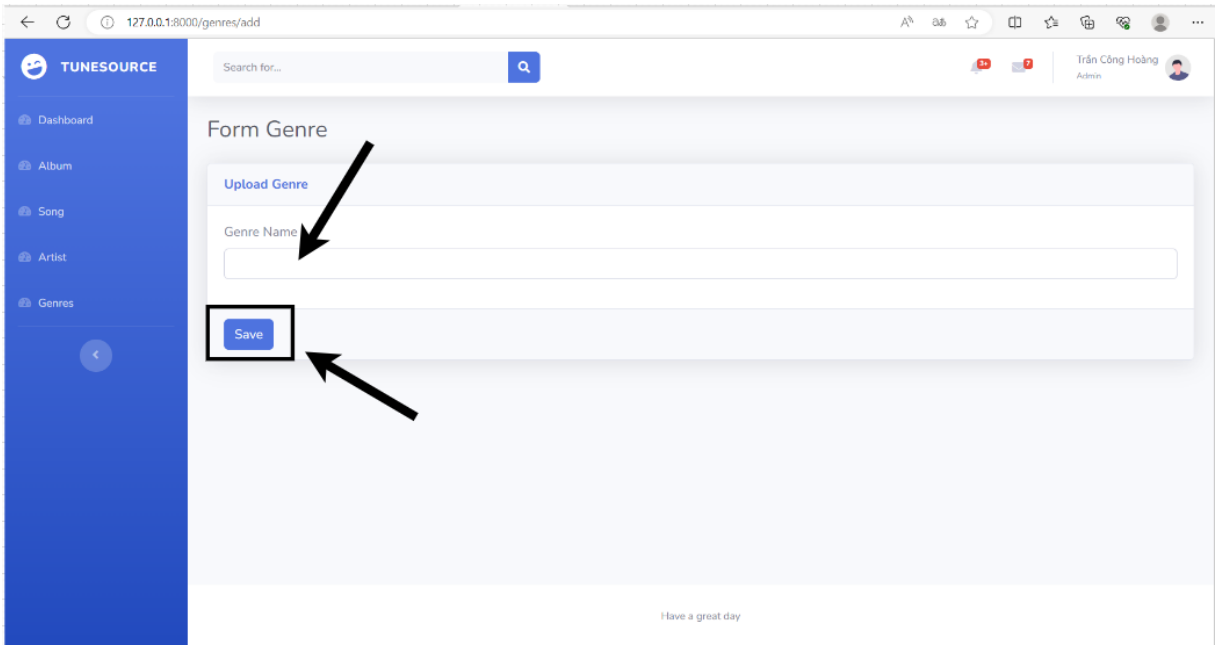


Figure 35: Add genre

- On the add new page, the admin will enter all the required information to add a new Genre.

b) Edit

To perform the function of editing information, the user clicks the Edit button on the right side of the editable information field.

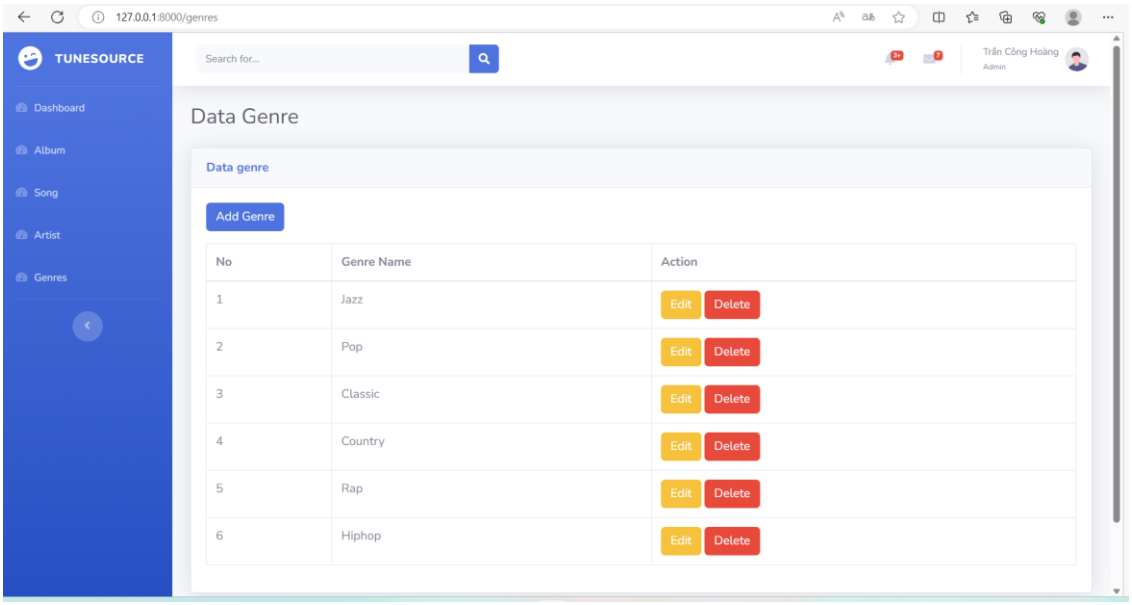


Figure 36: Edit genre

After clicking the Edit button, the user is navigated to the Edit page with the ID being the ID of the selected field.

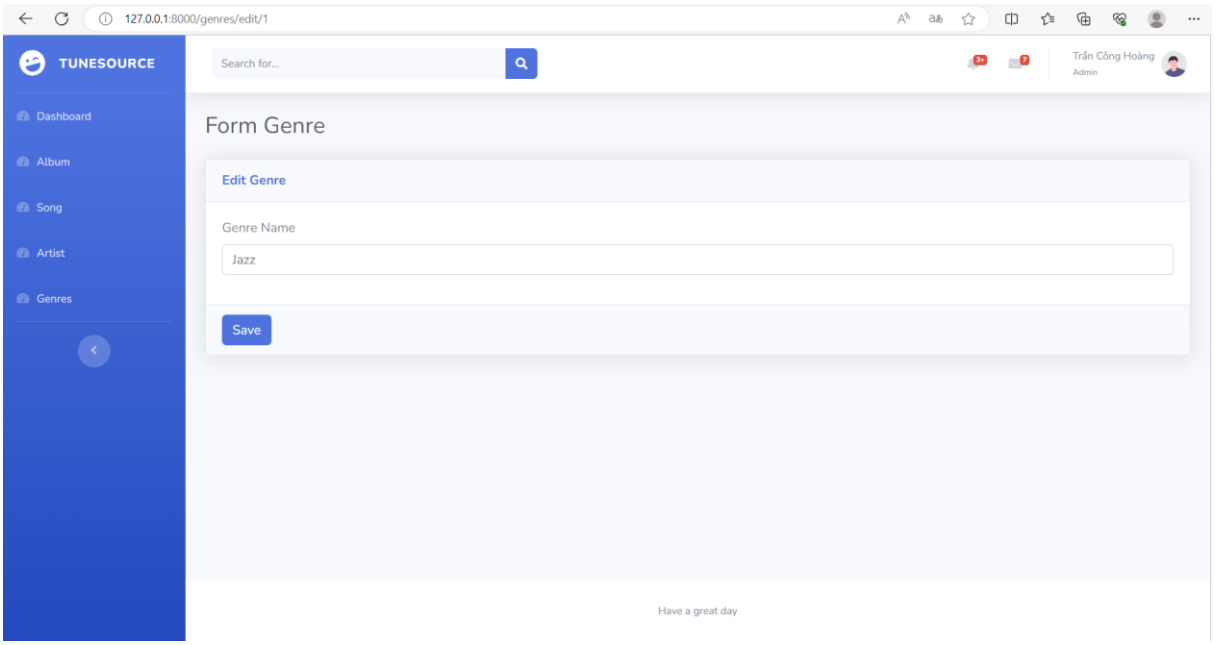


Figure 37: Edit genre

Here, the user will make changes to the information that needs to be edited

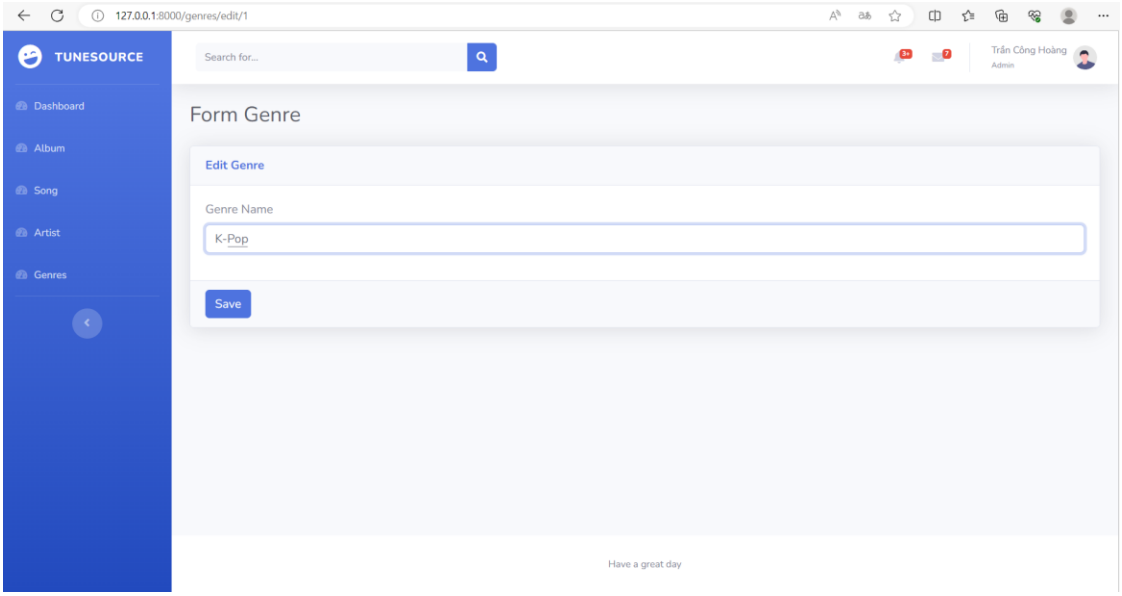


Figure 38: Edit genre

After successfully changing the information, the user presses the Save button to save the edited data

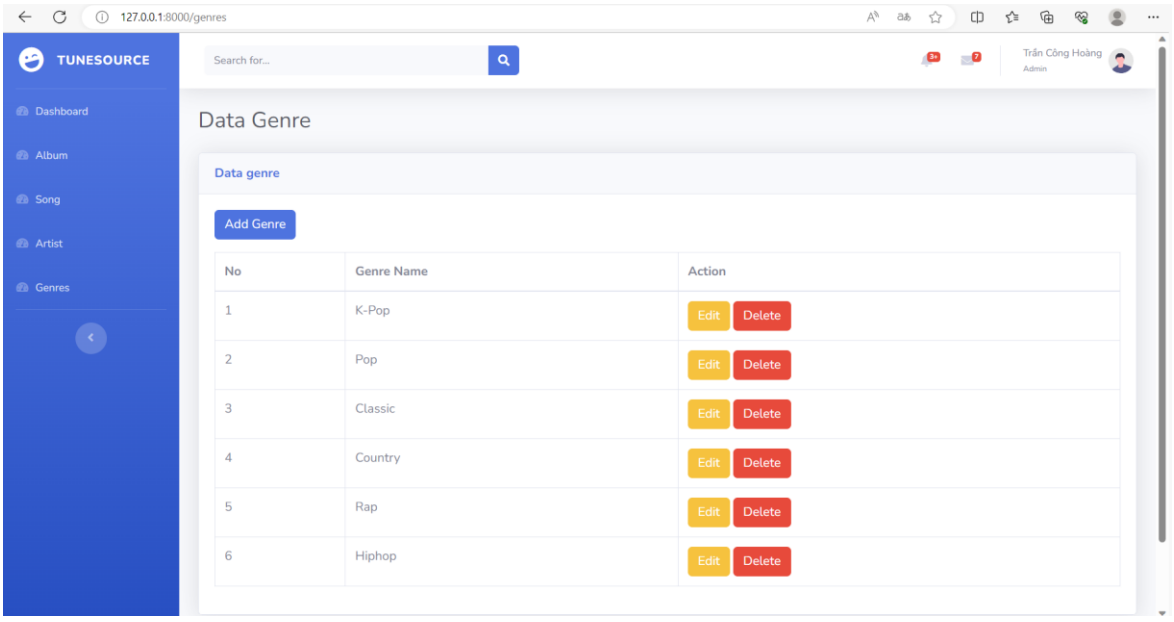


Figure 39: Edit genre success

c) Delete

If the save is successful, the user will return to the original page and the edited information will be displayed here.

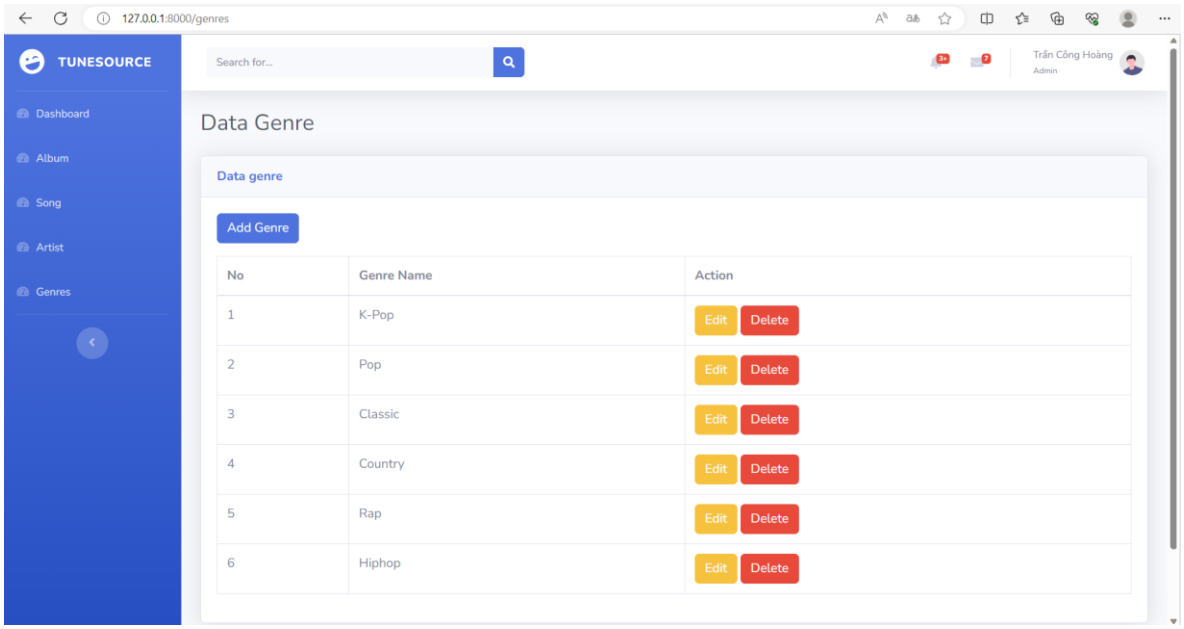


Figure 40: Delete genre

To perform the delete function, the user clicks the Delete button to the right of the row of information to be deleted.

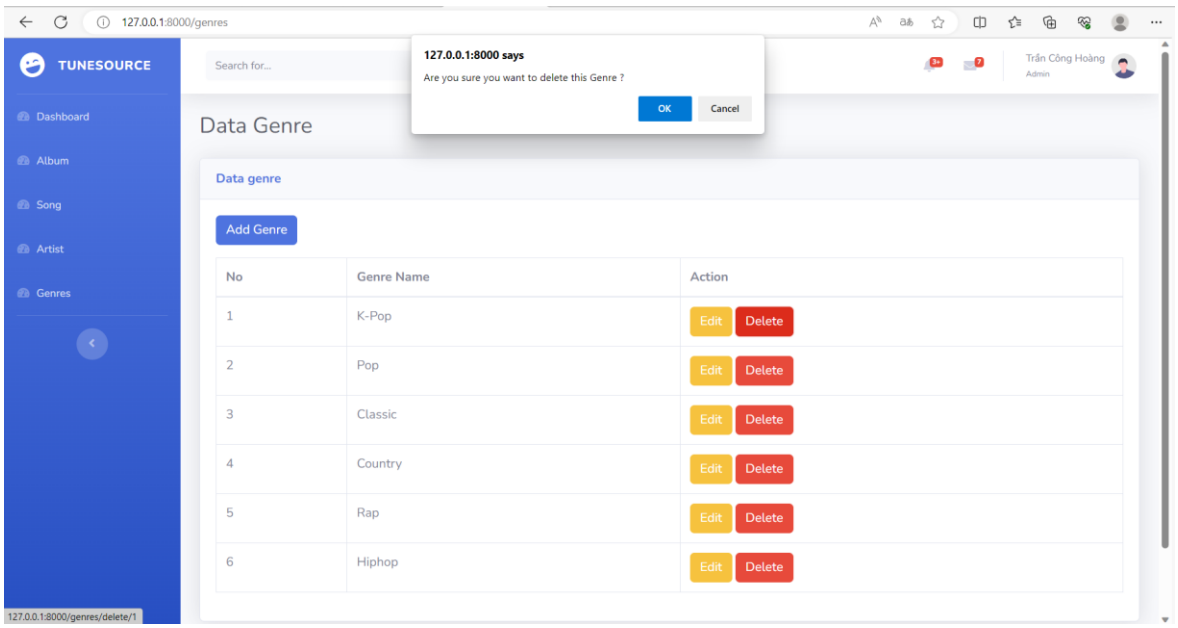


Figure 41: Delete genre

4) Instructions for adding, editing and deleting albums

a) Add

- Click on the Album icon in the left navigation bar of the screen.

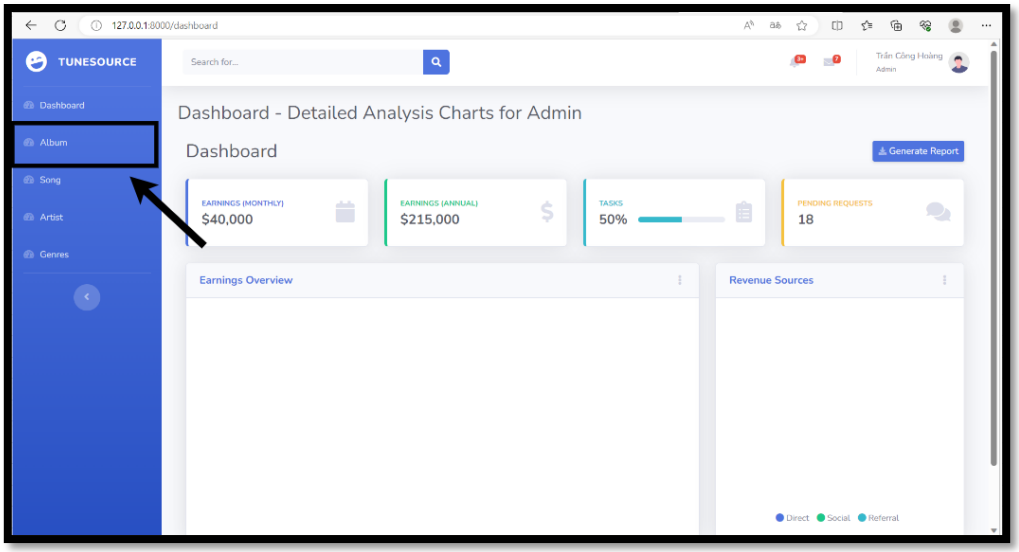


Figure 42: Web album

- In Album, to add new album, admin will click on Add Album button.

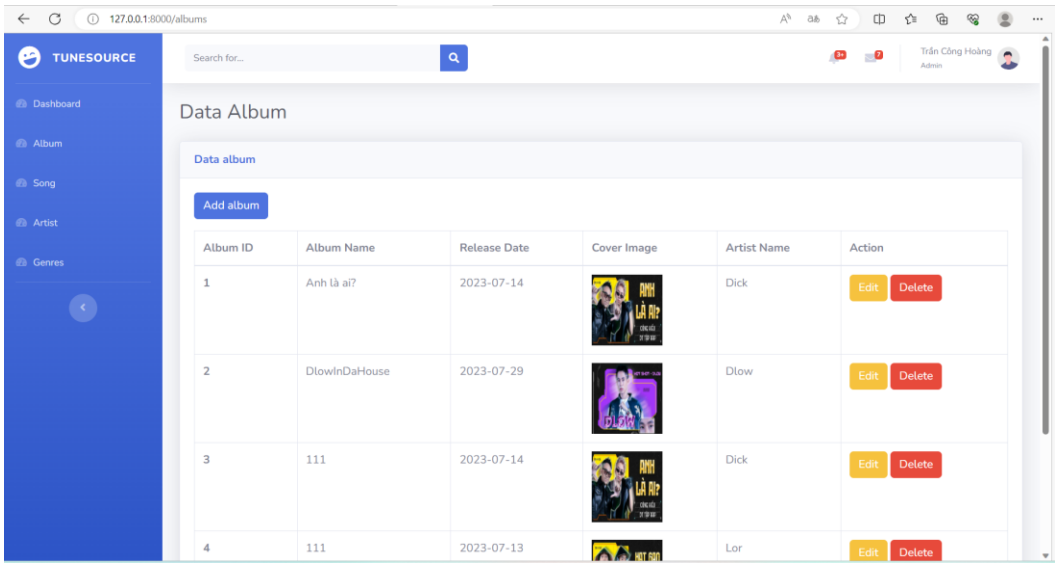


Figure 43: Add Album

- After clicking the Add album button, the admin will be redirected to the page to add a new Album.

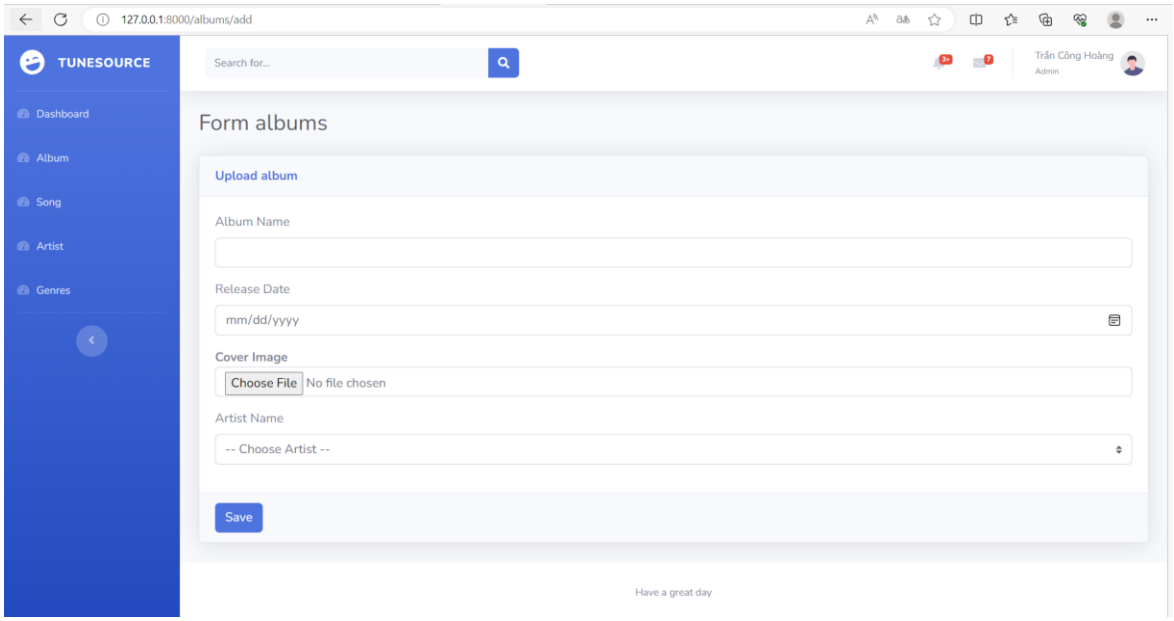


Figure 44: Add album

- On the new page, the admin will enter all the required information to add a new album.

- In the Image field, the admin will enter by clicking the Chose File button to add an image file, this field requires an image file and has the format of png,jpg,jpeg.

- After entering all the information, press the Save button to save the data and will be redirected to the Album page, the newly added data will be displayed here.

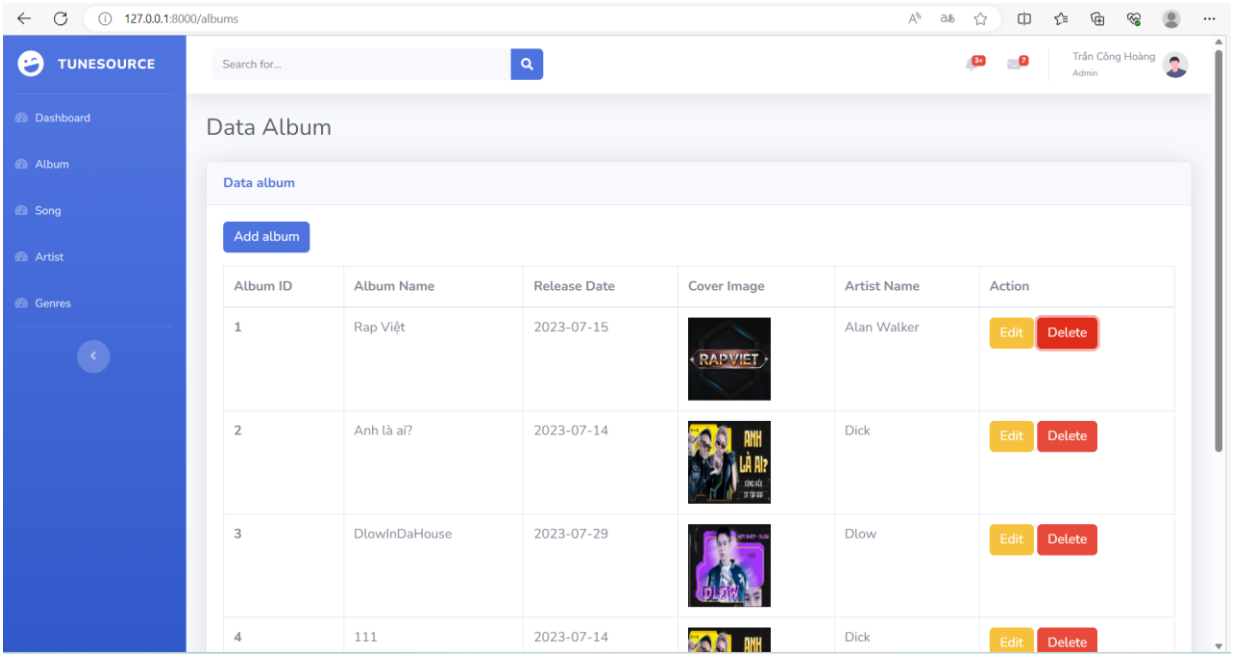


Figure 45: After add album

b) Edit

To perform the function of editing information, the user clicks the Edit button on the right side of the editable information field.

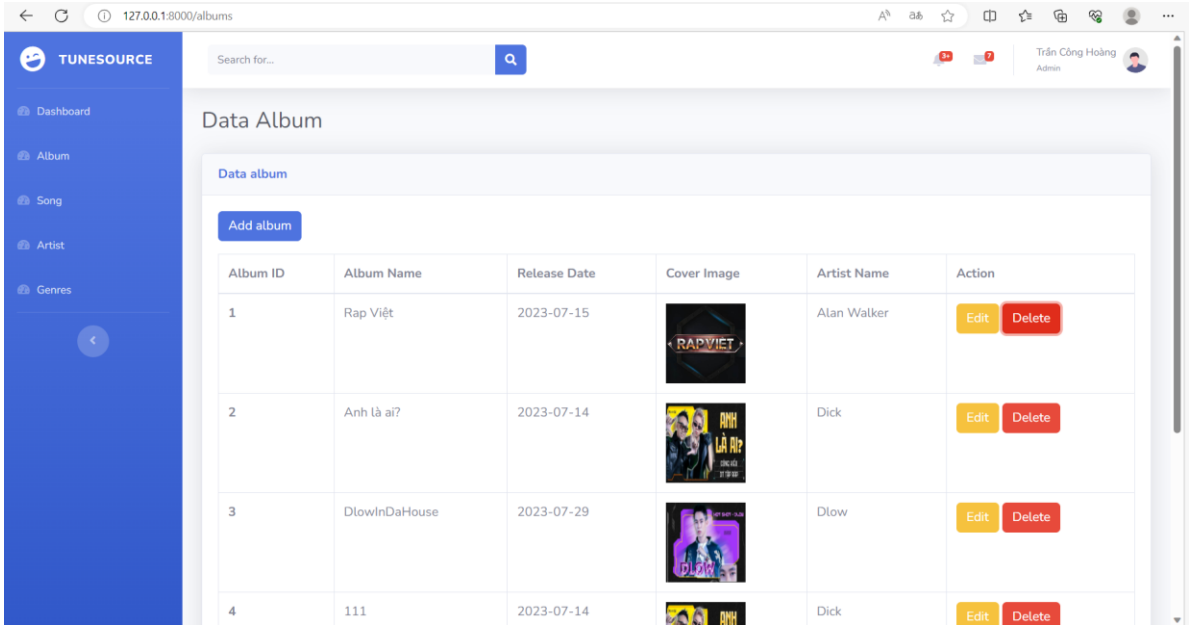
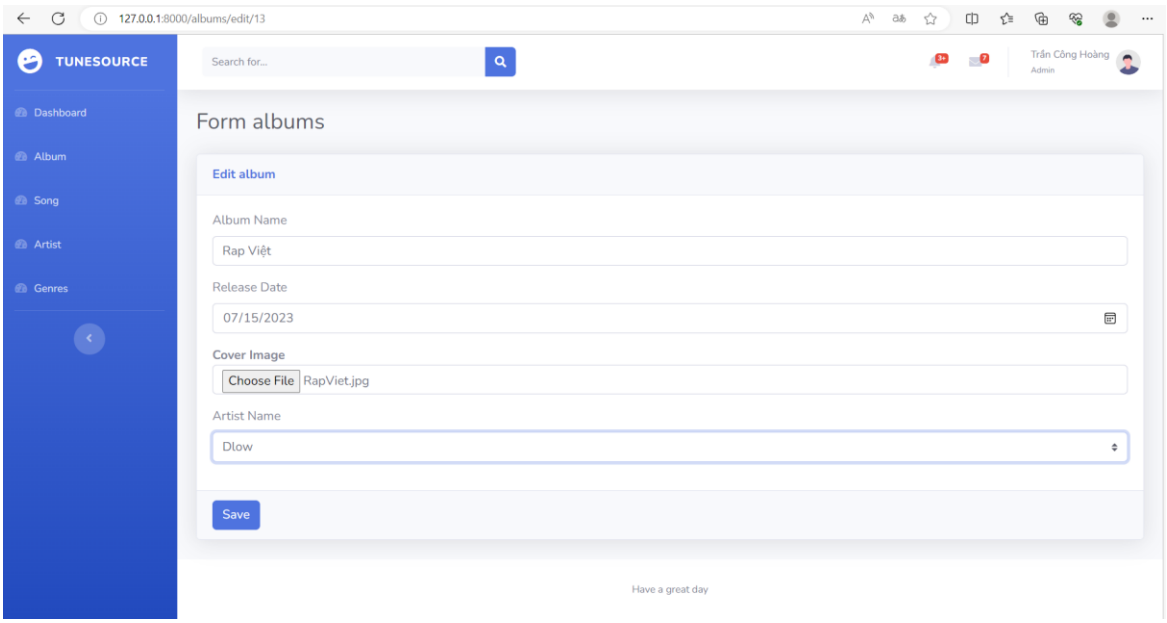


Figure 46: Edit album

After clicking the Edit button, the user is navigated to the Edit page with the ID being the ID of the selected field.



Here, the user will make changes to the information that needs to be edited

After successfully changing the information, the user presses the Save button to save the edited data.

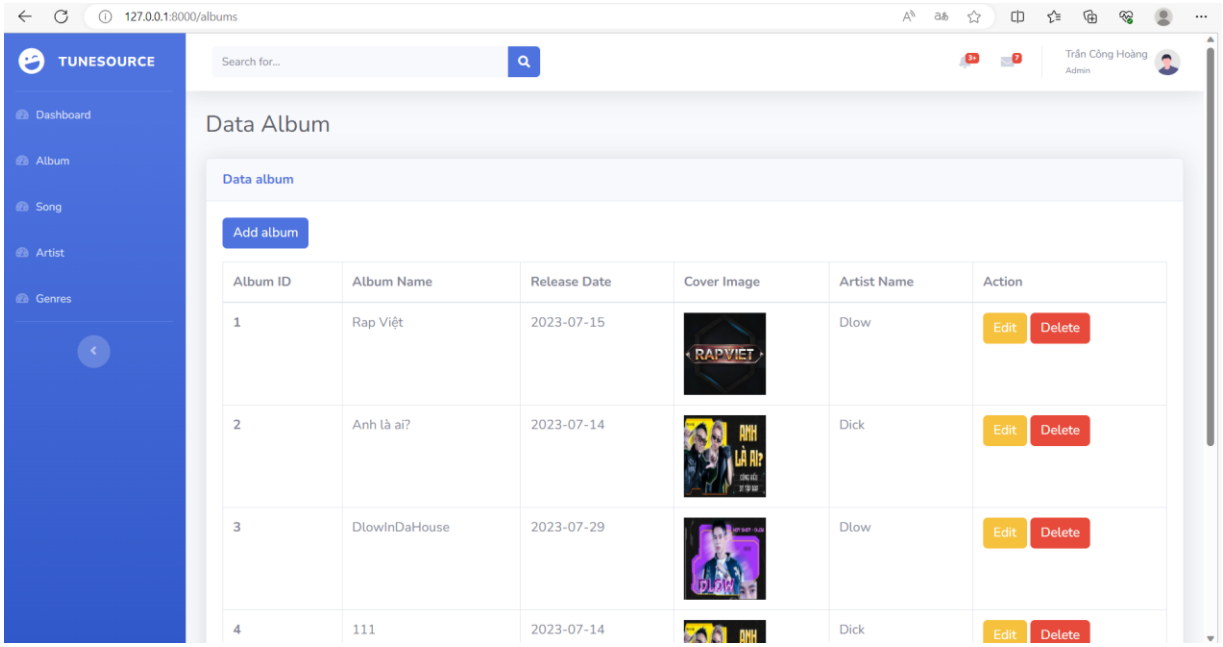


Figure 47: Edit success

c) Delete

If the save is successful, the user will return to the original page and the edited information will be displayed here.

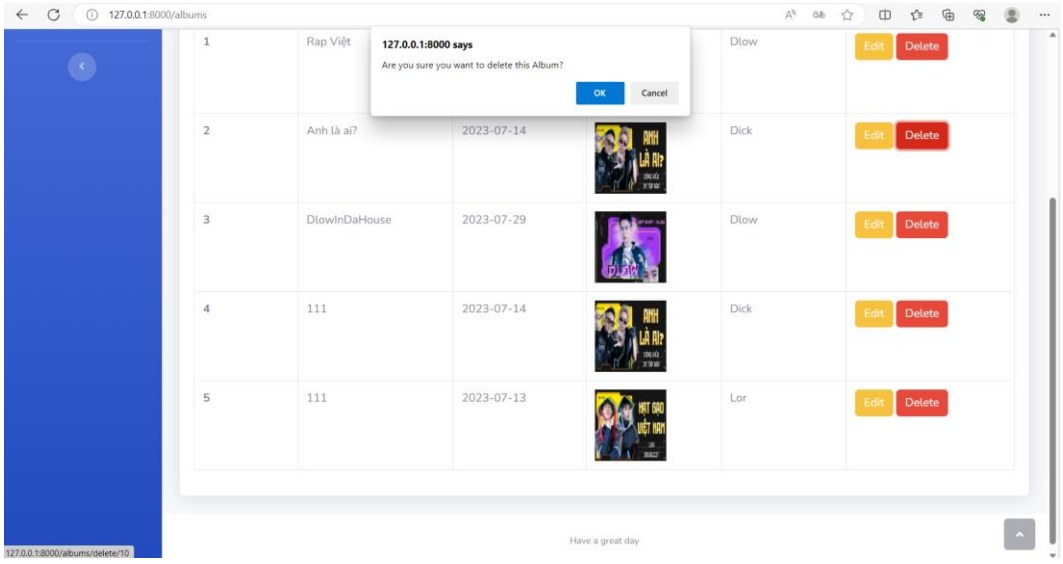


Figure 48: Delete album

To perform the delete function, the user clicks the Delete button to the right of the row of information to be deleted.

5) Instructions for adding, editing and deleting songs

a) Add

- To perform Song management, click on the Song icon in the left navigation bar of the screen.

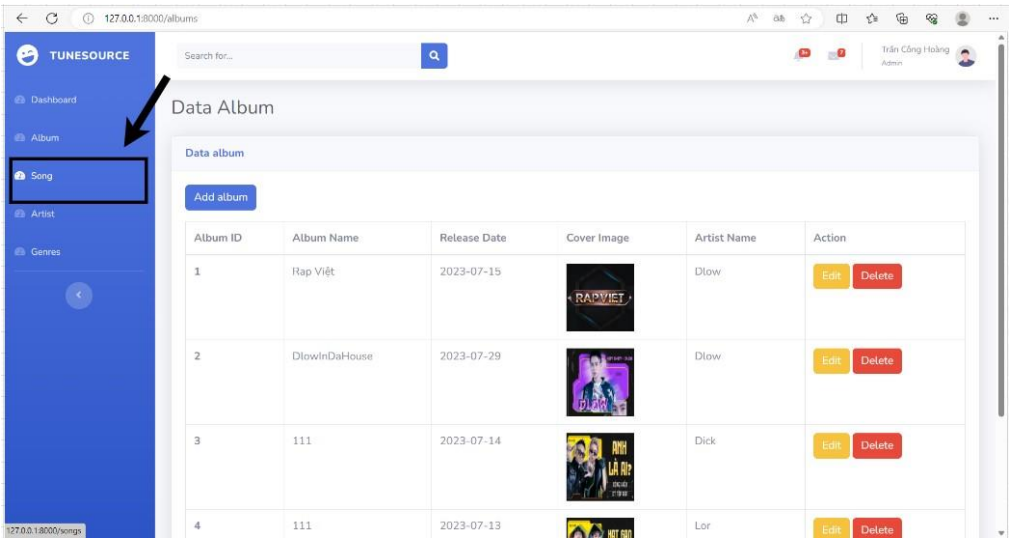


Figure 49: Add song

- In Song, to add a new Song, admin will click on the Add Song button.

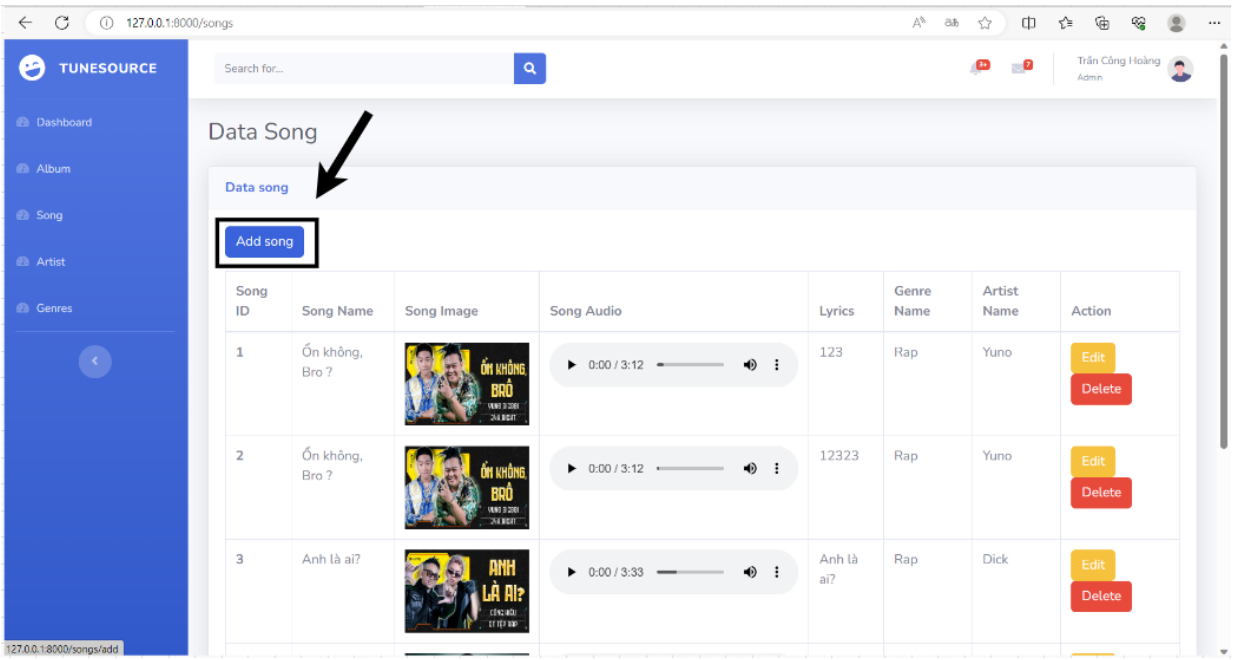


Figure 50: Add song

- After clicking the Add Song button, the admin will be redirected to the page to add a new Song.

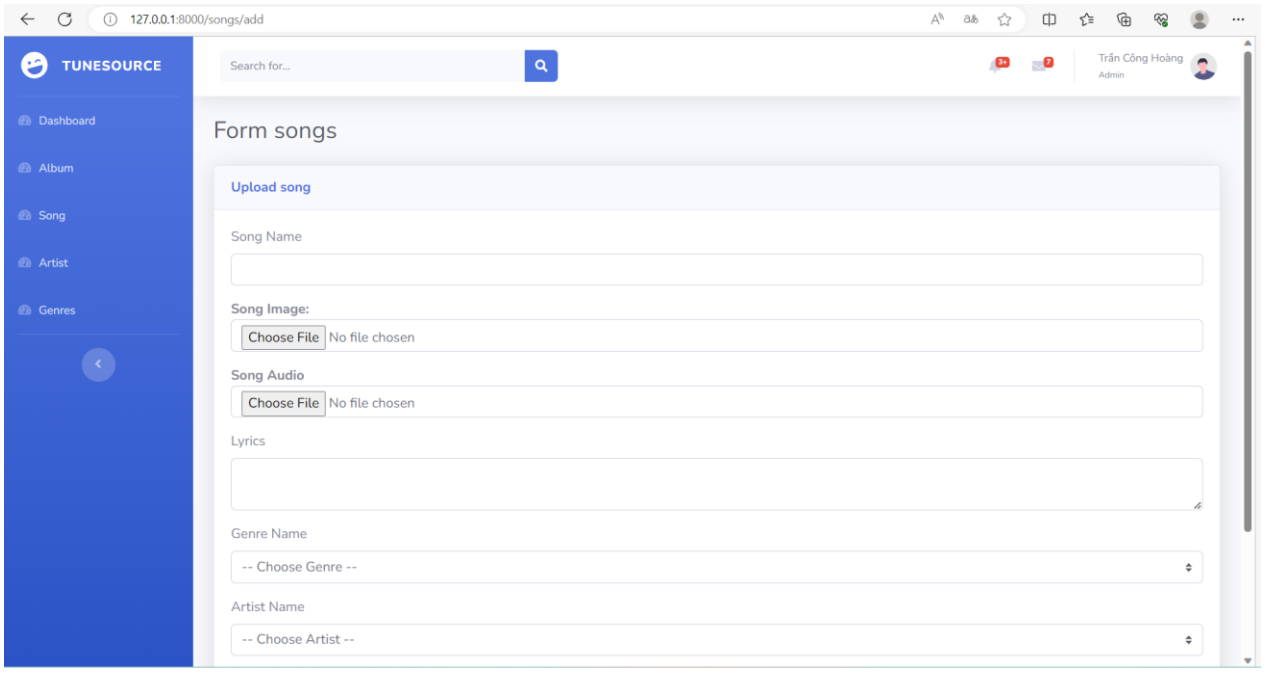


Figure 51: Add song

- On the new page, the admin will enter all the required information to add a new Song.

- In the Song Image and Audio fields, the admin will enter by clicking the Chose File button to add image files and audio files, this field requires image files and has the format of png,jpg,jpeg and formatted audio files. is mp3,wav,ogg.

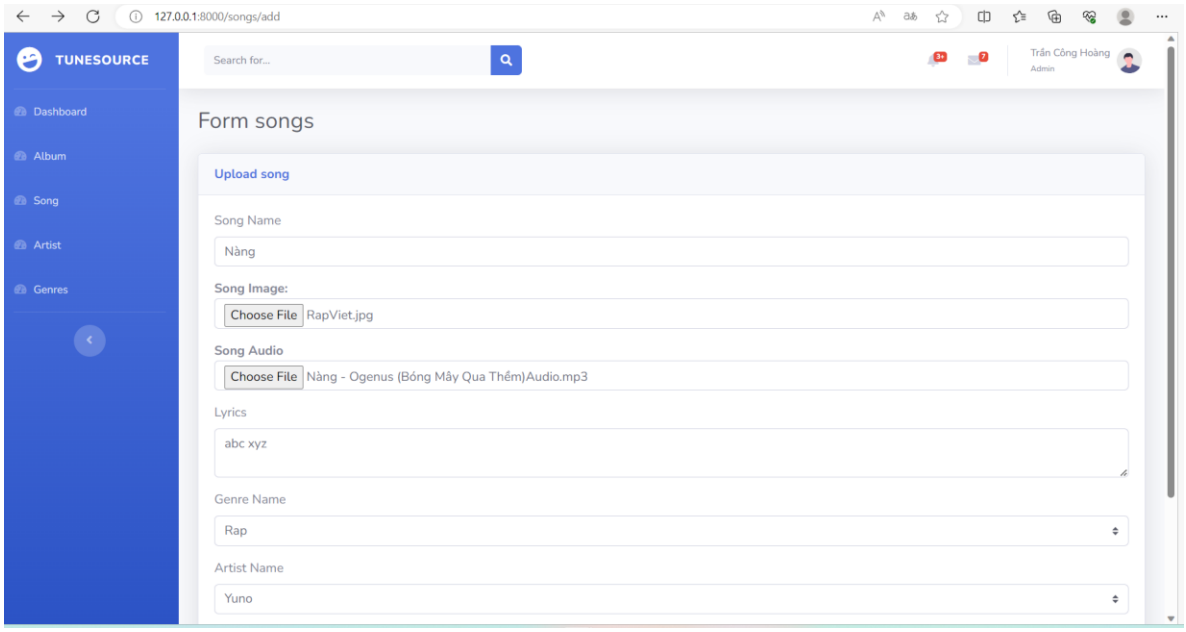


Figure 52: Input data

- After entering all the information, click the Save button to save the data.

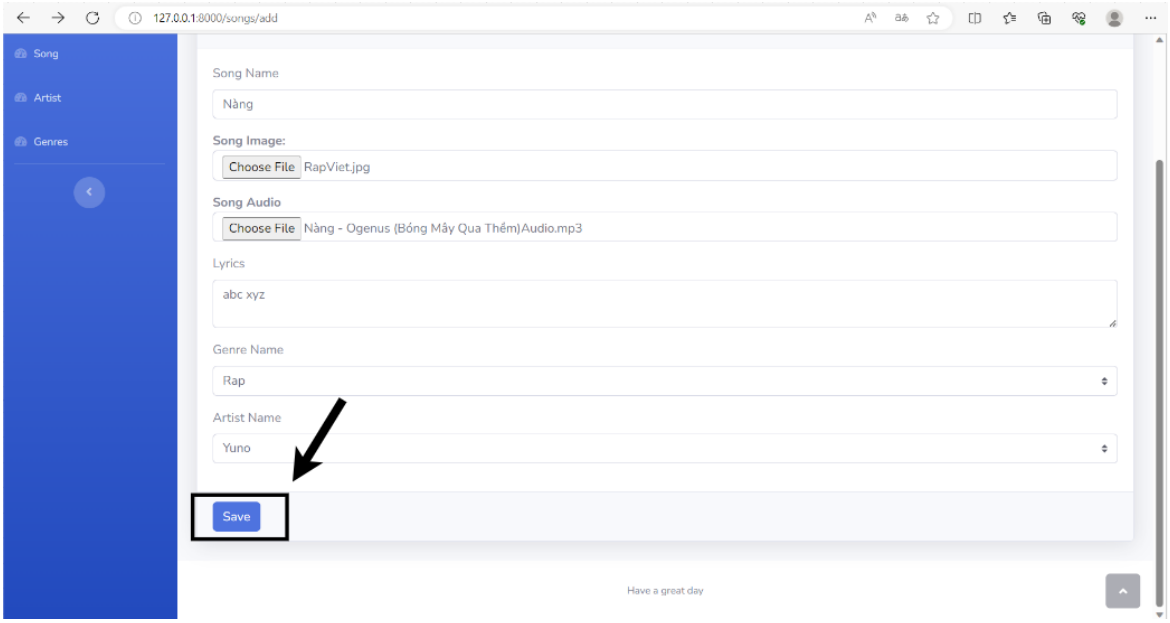


Figure 53: Save song

- Successfully saved will go to the initial management page, the newly added data will be displayed on the top of the page.

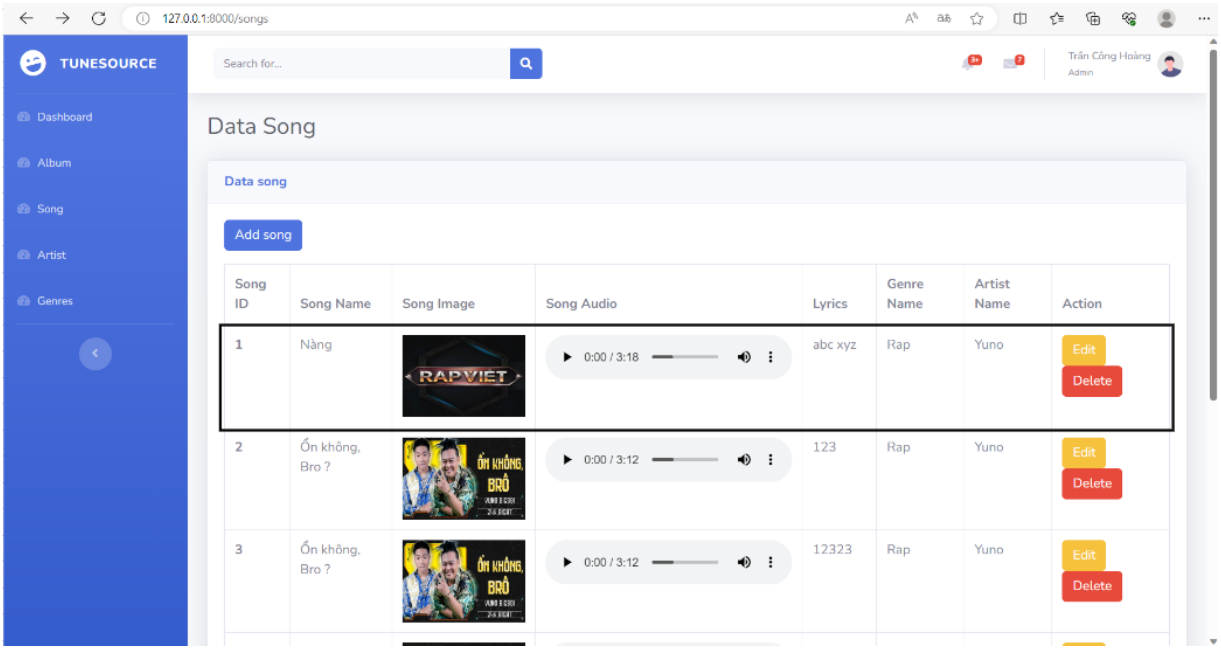


Figure 54: Add song success

b) Edit

To perform the function of editing information, the user clicks the Edit button on the right side of the editable information field.

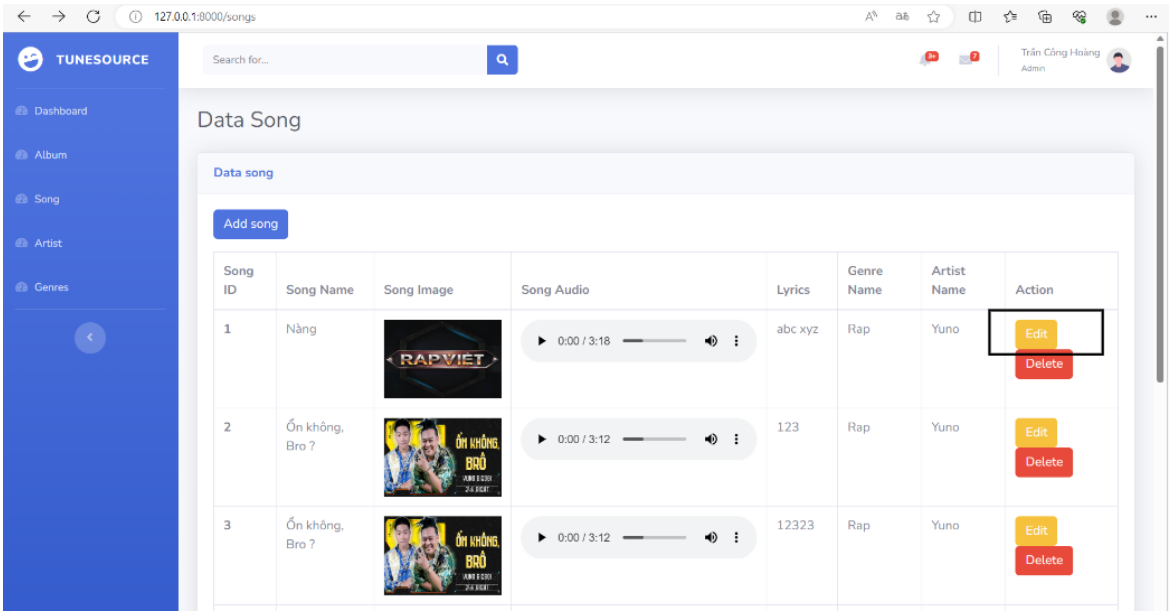


Figure 55: Button edit song

After clicking the Edit button, the user is navigated to the Edit page with the ID being the ID of the selected field.

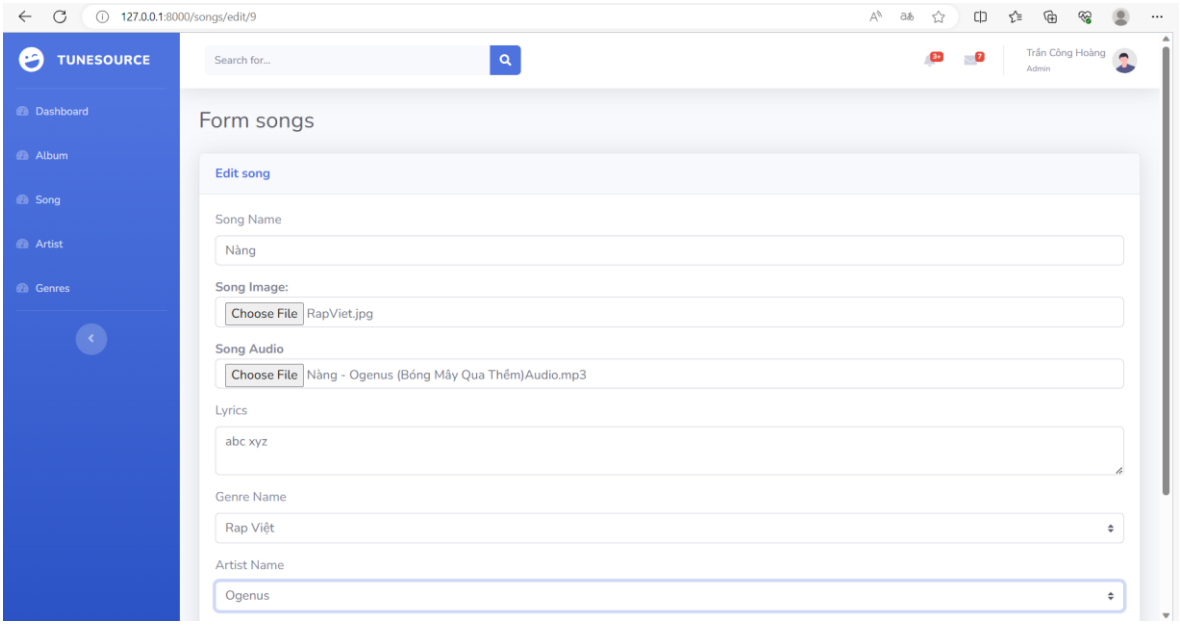


Figure 56: Edit web

Here, the user will make changes to the information that needs to be edited:

After successfully changing the information, the user presses the Save button to save the edited data:

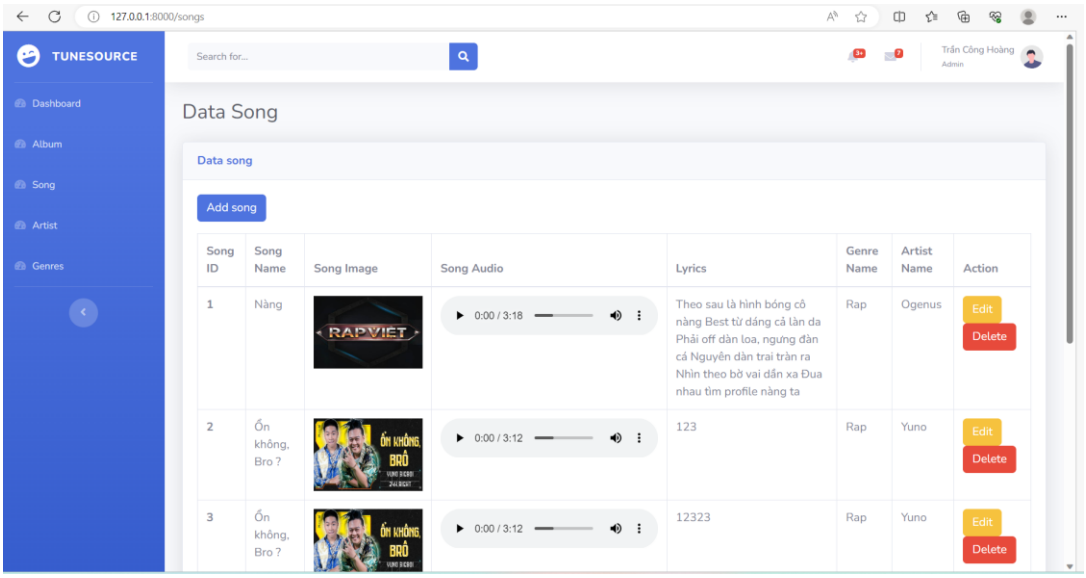


Figure 57: Edit success

c) Delete

If the save is successful, the user will return to the original page and the edited information will be displayed here.

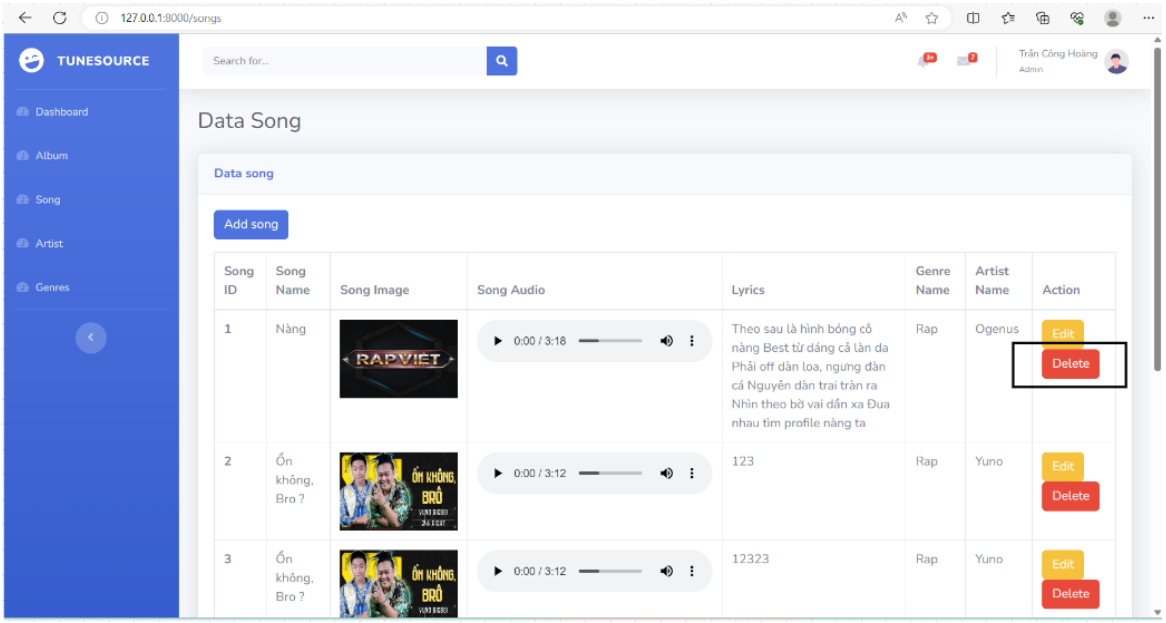


Figure 58: Delete song

To perform the delete function, the user clicks the Delete button to the right of the row of information to be deleted.

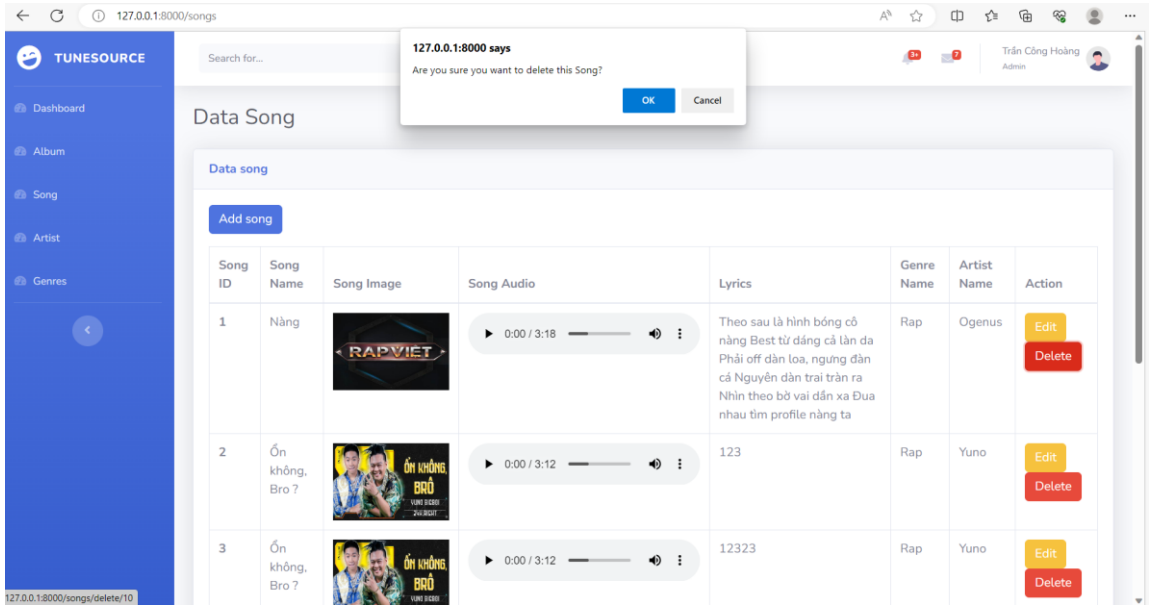


Figure 59: Warning delete

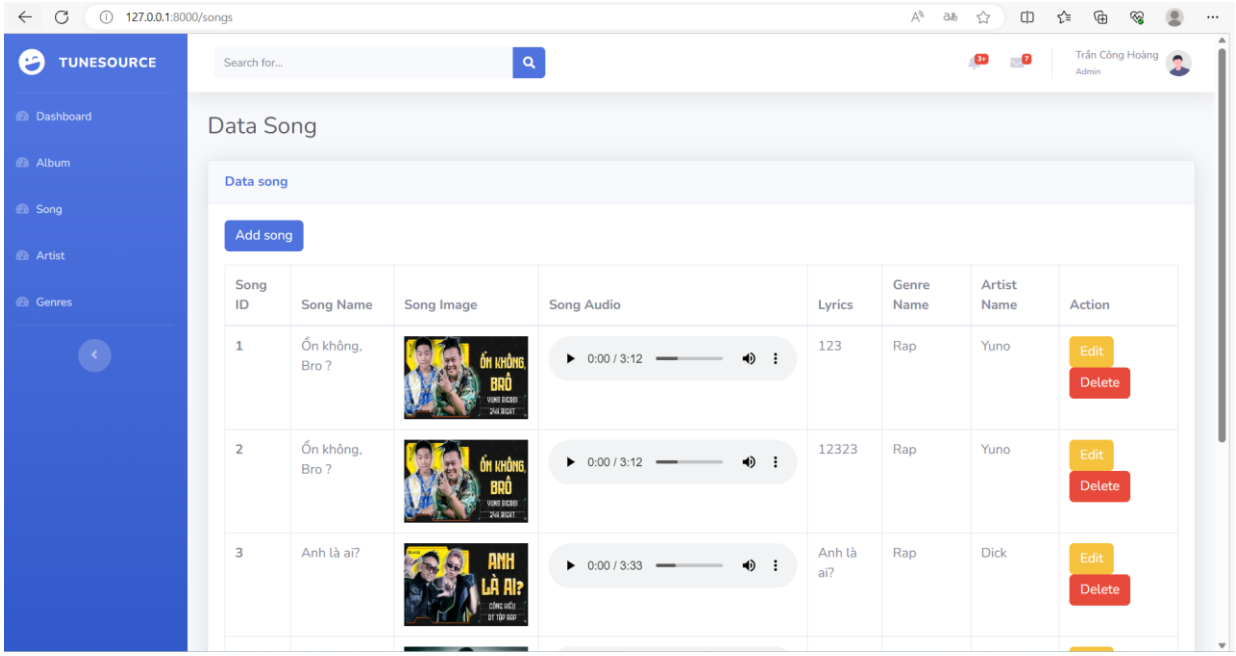


Figure 60: After delete

6) Logout

To log out, users press the icons in the upper right corner of the screen.

After clicking, a menu navigation bar will appear, the user presses the Logout button to log out.

Logout successful, user will be back in login.

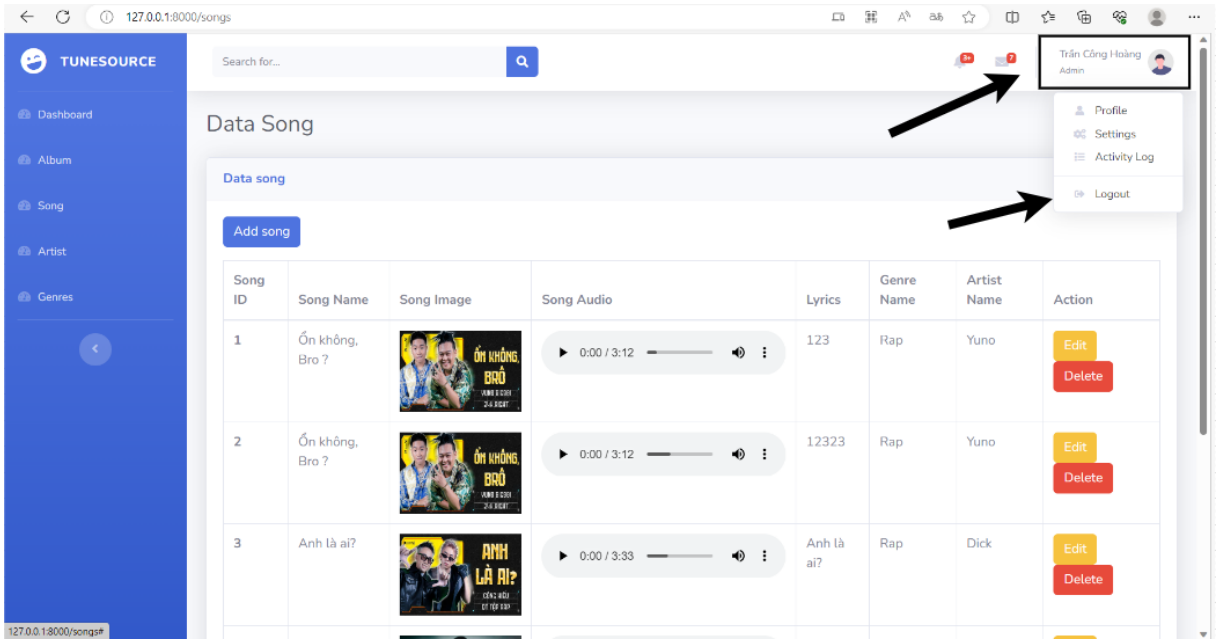


Figure 61: Logout

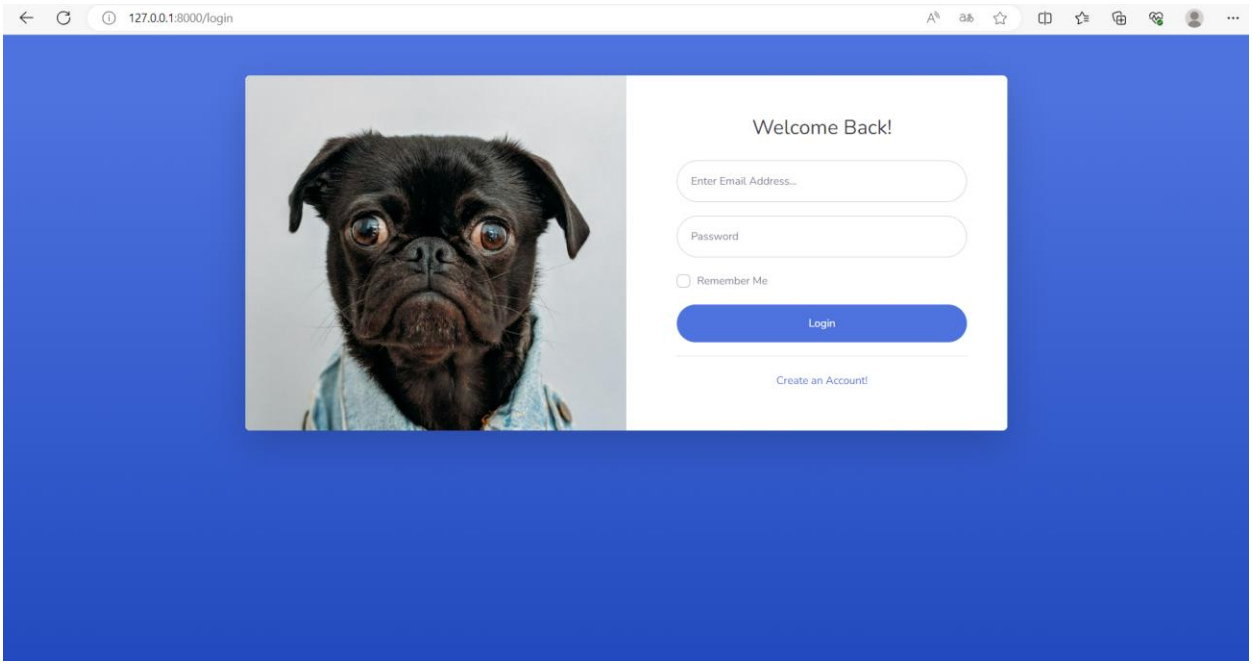


Figure 62: Logout success

C. Conclusion.

In this report, I have continued our analysis of Tune Source's project to create the capability of selling digital music downloads to customers through kiosks in their stores and over the Internet using their website. I have divided the report into several parts, each addressing a specific aspect of the project.

In Part P5, I discussed the software investigation undertaken to meet Tune Source's business needs. I identified the business need and gathered information from stakeholders such as customers, employees, and management using techniques such as interviews, surveys, and focus groups. We also analyzed existing documentation and data to identify the business needs and requirements.

In Part P6, I described the use of appropriate software analysis tools and techniques to carry out the software investigation and create supporting documentation. I documented requirements using tools such as user stories, use cases, and functional requirements, and created diagrams such as Use Case diagrams, Entity-Relationship diagrams (ERD), and Data Flow diagrams (DFD) to help visualize and understand the system that was going to be implemented.

In Part P7, I explained how user and software requirements have been addressed. We used methods such as software behavioral specification methods to model the behavior of the system and show how it meets the requirements. I also evaluated the reliability and effectiveness of the software by conducting tests such as unit testing, integration testing, and system testing.

Overall, I have provided a comprehensive and detailed analysis of Tune Source's project to create the capability of selling digital music downloads to customers through kiosks in their stores and over the Internet using their website. I hope that our findings will be useful to Tune Source in implementing this system and meeting their business needs.

D. References

(1) GeeksforGeeks. (n.d.). Types and Components of Data Flow Diagram (DFD). Retrieved from https://www.geeksforgeeks.org/types-and-components-of-data-flow-diagram-dfd/
(2) Lucidchart. (n.d.). Data Flow Diagram. Retrieved from https://www.lucidchart.com/pages/data-flow-diagram
(3) Lucidchart. (n.d.). Data Flow Diagram Symbols. Retrieved from https://www.lucidchart.com/pages/data-flow-diagram/data-flow-diagram-symbols
(4) StudyCorgi. (2022, May 24). Data Flow Diagram: Components and Guiding Rules. Retrieved from https://studycorgi.com/data-flow-diagram-components-and-guiding-rules/
(5) Quora. (n.d.). What are the components of data flow diagram? Retrieved from https://www.quora.com/What-are-the-components-of-data-flow-diagram
(6) Lucidchart (2023) ER Diagram (ERD) - Definition & Overview. Available at: https://www.lucidchart.com/pages/er-diagrams. (Accessed: 8 January 2023).
(7) GeeksforGeeks (n.d.) Introduction of ER Model. Available at: https://www.geeksforgeeks.org/introduction-of-er-model/. (Accessed: 8 January 2023).
(8) Database Star (n.d.) A Guide to the Entity Relationship Diagram (ERD). Available at: https://www.databasestar.com/entity-relationship-diagram/. (Accessed: 8 January 2023).
(9) Moqups (n.d.) ERD Templates | ER Diagram Examples. Available at: https://moqups.com/templates/diagrams-flowcharts/erd/. (Accessed: 8 January 2023)
(10) Lucidchart (2023) ER Diagram (ERD) - Definition & Overview. Available at: https://www.lucidchart.com/pages/er-diagrams. (Accessed: 8 January 2023).
(11) Database Star (n.d.) A Guide to the Entity Relationship Diagram (ERD). Available at: https://www.databasestar.com/entity-relationship-diagram/. (Accessed: 8 January 2023).
(12) Creately (n.d.) What is an Entity Relationship (ER) Diagram. Available at: https://creately.com/guides/er-diagrams-tutorial/. (Accessed: 8 January 2023).
(13) EDUCBA (n.d.) Complete Guide to Entity Relationship Model. Available at: https://www.educba.com/entity-relationship-model/. (Accessed: 8 January 2023).
(14) BeginnersBook (2015) Entity Relationship Diagram - ER Diagram in DBMS. Available at: https://beginnersbook.com/2015/04/e-r-model-in-dbms/. (Accessed: 8 January 2023).