Database Driven Web Application

COMP1048: Databases and Interfaces (2024-2025)

Matthew Pike & Yuan Yao

Table of contents

Document Revision History

We will maintain a record of changes to this document here.

Version	Date	Description
1.0	2024-12-02	Initial version of the task sheet.

Table 1: Document Revision History.

1 Overview

The iMusic Scenario is Fictitious

Please note that this assignment is based on a fictional scenario, and the company mentioned does not exist. Furthermore, the data provided may not be accurate or complete.

iMusic is a company that sells music records. Previously, the company operated a physical store in the city centre but has recently transitioned to an online-only business model. To support this change, they plan to update their website.

iMusic initially hired a professional developer to carry out this work, but due to other commitments, the developer was unable to complete it. You have now been hired to finish the project.

iMusic has provided a list of website requirements outlined in the following sections. The website is partially implemented, and your task is to complete its development. To complete the assignment, you will need to modify only the `iMusic.py` file.

The website uses the following technologies:

- HTML + Jinja2: For the website's front-end. The previous developer has already created the necessary web pages using HTML and Jinja2 templates. **You should not modify these templates.**

- Python + Flask: For the website's back-end. The website is built using the Flask web application framework. You will need to complete the Python functions in the `iMusic.py` file to implement the required functionality. The iMusic team use Python 3.12. During testing of your solution, the iMusic team will use the 3.0.3 version of Flask.

- SQLite: For the website's database. The website stores its data in an SQLite database. You will need to interact with the database using the `sqlite3` module in Python. Do not use any other database libraries or modules such as SQLAlchemy or Pandas. During the testing of your solution, the iMusic team will use the 3.44.0 version of SQLite.

  - Please note: The database schema is identical to the database used in the DBI Quiz. Some of the data, however, has been modified for this assignment. The original source of the database is the Chinook Database.

You are not permitted to use any other technologies or import any additional modules beyond those already imported in the `iMusic.py` file or those provided by the Python standard library.

2 Tasks

2.1 Task 1: Fixing Customer Data (5 Marks)

2.1.1 Task Brief

Some customer telephone and fax numbers in the database are incorrect. You have been provided with an example Tab-seperated values (TSV) file containing the correct information for these customers, which you will use to update the database. To do this, modify the `update_customers` function in the `iMusic.py` file. This function will be automatically triggered when a user uploads a TSV file via the form on the `/upload/` route of the website.

The uploaded file will be a TSV file containing the following columns:

- `CustomerId`: The ID of the customer to be updated (**required**).
- `Phone`: The new telephone number for the customer (**required**).
- `Fax`: The new fax number for the customer (**optional**).

The `CustomerId` and `Phone` fields will always be provided in the TSV file, while the `Fax` field is optional.

- If a value is included in the `Fax` column, update the fax number for the customer.
- If the `Fax` column is missing, leave any existing fax number unchanged. Do not delete the existing fax number.

An example TSV file has been provided for you to use in testing your solution. The file is named `original_customers.tsv` and is located in the `data` directory.

To parse the TSV file, you should use the `csv` module, which is part of the Python standard library. The `csv` module provides functionality for reading and writing CSV and TSV files. You can refer to the csv module documentation for more information. Reading and comprehending the documentation is an essential skill for developers, and an assessed skill in this assignment.

Ensure that each `CustomerId` exists in the database before updating. If a `CustomerId` does not exist in the database, ignore that record in the TSV file. You do not need to validate the format of the telephone or fax numbers. Only the specified customers in the TSV file should be updated; no other records should be affected.

Once the customer data has been updated, they will be redirected back to the index page (‘/’). If there are any errors during the update process, there is no need to display an error message to the user; simply redirect them back to the index page. Your solution should handle any errors gracefully and not crash the server.

2.1.2 Evaluation Procedure

When testing your solution, the iMusic team will:

1. Navigate to the `/upload/` route on the website.
2. Upload a TSV file containing updated customer data. Several TSV files with different customer data will be used in testing; these files will not be provided to you in advance. Ensure your solution can handle any valid TSV file, not just the example provided. You are encouraged to test your solution with different TSV files you create to ensure it works correctly.

3. Submit the form to trigger the `update_customers` function.
4. Verify that the customer data in the database has been correctly updated.

2.1.3 Constraints

Your implementation must meet the following requirements:

- The entire implementation must be contained within the `update_customers` function. Do not create additional functions or modify the template file.
- The implementation must run as a Flask application, using the command `python iMusic.py` or `python3 iMusic.py`.
- Use the `sqlite3` module exclusively for database interactions. Other modules or approaches (e.g., SQLAlchemy) are not permitted.
- Use the `csv` module for reading the TSV file. Refer to the csv module documentation as needed. Other modules or approaches (e.g., Pandas) are not allowed.

2.1.4 Marking Scheme

Task 1 is worth 5 of the 25 marks available for the assignment. The following criteria will be used to assess your work:

ID	Requirement	Details	Marks
RQ1_1	Correct File Reading	Successfully reads the TSV file using the `csv` module, correctly parsing the customer data.	1
RQ1_2	Database Connection	Establishes a connection to the SQLite database using the `sqlite3` module.	1
RQ1_3	Correct Data Handling	Correctly identifies whether a `CustomerId` exists in the database and updates the telephone/fax numbers	1
RQ1_4	Accurate Database Update	Accurately updates existing records in the database with the correct customer data, without affecting other records.	2

Table 2: Marking Criteria for Task 1.

2.2 Task 2: Customer Statistics Display (8 Marks)

2.2.1 Task Brief

The iMusic team requires an overview of basic customer statistics to enhance their understanding of the customer base and buying patterns across different countries. You have been tasked with implementing new functionality on the website that displays statistics for all customers in a selected country, as well as aggregated statistics for that country. Additionally, you will need to provide the option of viewing statistics for all customers, denoted by the country 'All'. To achieve this, you will need to complete the `statistics`, `get_all_countries` and `get_statistics` functions in the `iMusic.py` file.

When a user navigates to the `/statistics/` route, the `statistics` function will be triggered. This function should render the `statistics.html` template file, which is provided in the `templates` directory. The template file contains a form with a dropdown menu that allows users to select a country. The form is submitted using the POST method, and the selected country should be sent to the `/statistics/` route. By default, the dropdown should display the option 'All', which will display statistics for all customers in the database.

The `get_all_countries` function should be completed to retrieve a list of distinct countries from the `Customer` table in the database. The countries should be sorted in ascending alphabetical order, with the option 'All' appearing at the first in the list. The `statistics` function should call the `get_all_countries` function to retrieve the list of countries and pass this data to the template for rendering.

Upon selecting a country from the dropdown, the page should display the following statistics for each customer situated in that country:

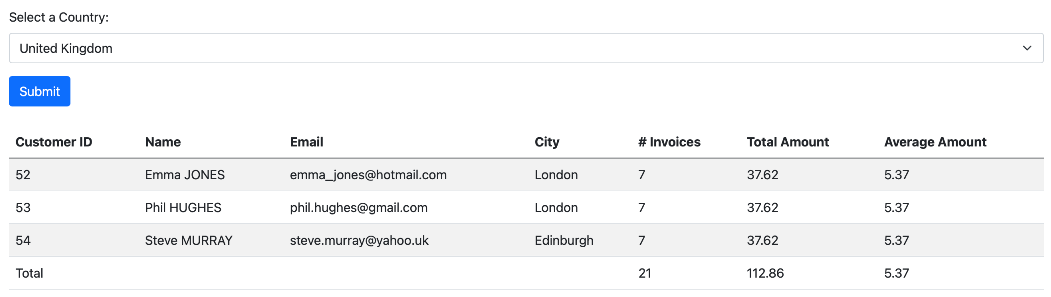
- Customer ID: The ID of the customer.
- Name: The full name of the customer (First Name + ' ' + LAST NAME).
  - Note: There is a single space between the first name and last name. The last name should be entirely in uppercase.
- Email: The customer's email address.
- City: The city of the customer.
- # Invoices: The total number of invoices in the database associated with the customer.
  - Note: All customers should be displayed, even if they have no invoices.
- Total Amount: The total sum of invoice values for the customer, rounded to two decimal places.
- Average Amount: The average invoice value for the customer, rounded to two decimal places.
  - If a customer has no invoices, the 'Total Amount' and 'Average Amount' should be displayed as 0.

Customers should be displayed in ascending order of their `CustomerId`.

In addition to the individual customer statistics, the page should also display the following aggregated statistics for the selected country at the bottom of the table:

- Total Customers: The total number of customers in the selected country.
  - This should have the text 'Total' in the 'Customer ID' column.

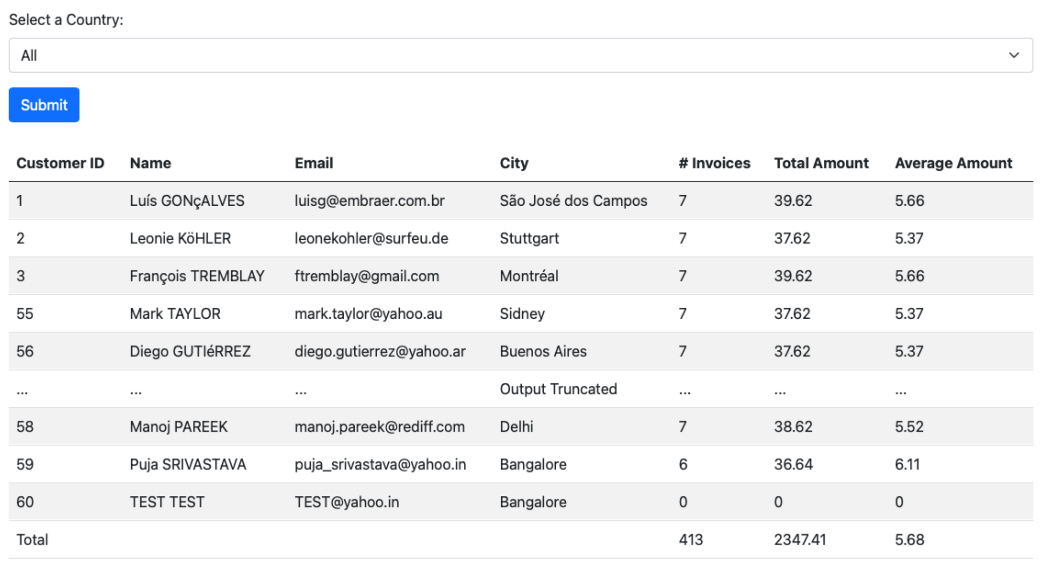
- **Total Invoices**: The total number of invoices in the database associated with customers in the selected country.
  - This value should be presented in the '# Invoices' column.
- **Total Amount**: The total sum of invoice values for customers in the selected country, rounded to two decimal places.
  - This value should be presented in the 'Total Amount' column.
- **Average Amount**: The average invoice value for customers in the selected country, rounded to two decimal places.
  - This value should be presented in the 'Average Amount' column.



Customer ID	Name	Email	City	# Invoices	Total Amount	Average Amount
52	Emma JONES	emma_jones@hotmail.com	London	7	37.62	5.37
53	Phil HUGHES	phil.hughes@gmail.com	London	7	37.62	5.37
54	Steve MURRAY	steve.murray@yahoo.uk	Edinburgh	7	37.62	5.37
Total				21	112.86	5.37

Figure 1: An example of the statistics page for a single country. The example shows the customer statistics for the country 'United Kingdom' and the aggregated statistics for that country at the bottom of the table.Please note that the actual data displayed may differ, and this is for illustrative purposes only.

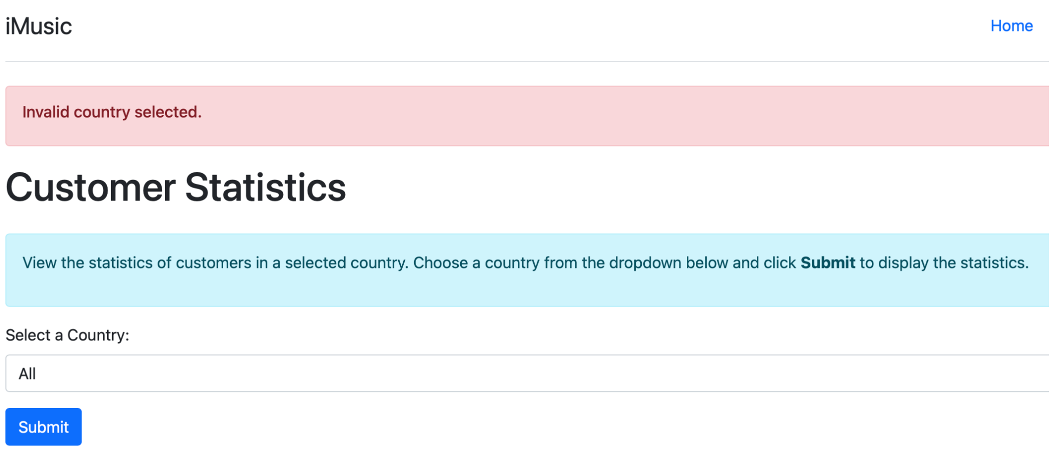
Your solution should handle the scenario where the user selects the 'All' option from the dropdown. In this case, the page should display the statistics for all customers in the database, as well as the aggregated statistics for all customers. The aggregated statistics for all customers should be displayed at the bottom of the table, similar to the aggregated statistics for a single country.



Customer ID	Name	Email	City	# Invoices	Total Amount	Average Amount
1	Luís GONçALVES	luisg@embraer.com.br	São José dos Campos	7	39.62	5.66
2	Leonie KöHLER	leonekohler@surfeu.de	Stuttgart	7	37.62	5.37
3	François TREMBLAY	ftremblay@gmail.com	Montréal	7	39.62	5.66
55	Mark TAYLOR	mark.taylor@yahoo.au	Sidney	7	37.62	5.37
56	Diego GUTIéRREZ	diego.gutierrez@yahoo.ar	Buenos Aires	7	37.62	5.37
...	...	...	Output Truncated	...	...	...
58	Manoj PAREEK	manoj.pareek@rediff.com	Delhi	7	38.62	5.52
59	Puja SRIVASTAVA	puja_srivastava@yahoo.in	Bangalore	6	36.64	6.11
60	TEST TEST	TEST@yahoo.in	Bangalore	0	0	0
Total				413	2347.41	5.68

Figure 2: An example of the statistics page for all customers. The example shows the customer statistics for all customers and the aggregated statistics for all customers at the bottom of the table. Please note that the actual data displayed may differ, and this is for illustrative purposes only.

Your solution must also handle scenarios where invalid country selections are provided. Please note that just because we specify the countries and values that **should** appear in the dropdown, we cannot trust the data received from the user. It is possible for a user to manipulate the HTML and send a request with a country that is not present in the dropdown. To ensure our application is secure and robust, we must handle this scenario appropriately. In such cases, the page should redirect the user back to the `/statistics/` route with the message 'Invalid country selected' and an alert type of `danger`. An example of how to do this is provided in the provided `iMusic.py` file. The message should be displayed in the same format as the error messages in the provided template file, making use of the `flash` function in Flask. An example of the error message is shown below:


iMusic

Home

Invalid country selected.

Customer Statistics

View the statistics of customers in a selected country. Choose a country from the dropdown below and click **Submit** to display the statistics.

Select a Country:

All

Submit

Figure 3: An example of the error message displayed when an invalid country is selected.

2.2.2 Evaluation Procedure

When testing your solution, the iMusic team will:

1. Navigate to the `/statistics/` route on the website.
2. Select a country from the dropdown menu and press `Submit`.
3. Verify that the customer statistics for the selected country are displayed correctly.
4. Verify that the aggregated statistics for the selected country are displayed accurately.

To further test your solution, the iMusic team will add new customers to the database and check that the statistics page updates correctly. Ensure that your solution can handle any valid data in the database, not just the examples provided. You are encouraged to test your solution with different data to ensure it works correctly—you will need to create your own test data for this task. The iMusic team will not provide you with additional test data.

Similarly, the iMusic team will attempt to access the `/statistics/` route with an invalid country selection to verify that the error message is displayed correctly.

2.2.3 Constraints

Your implementation must meet the following requirements:

- The entire implementation must be contained within the `statistics`, `get_all_countries`, and `get_statistics` functions. Do not create additional functions or modify the template file.

- The implementation must run as a Flask application, using the command `python iMusic.py` or `python3 iMusic.py`.
- Use the `sqlite3` module exclusively for database interactions. Other modules or approaches (e.g., SQLAlchemy) are not permitted.
- Do not modify the provided template file (`templates/statistics.html`).

2.2.4 Marking Scheme

Task 2 is worth 8 of the 25 marks available for the assignment. The following marking criteria will be used to assess your work:

ID	Requirement	Details	Marks
RQ2_1	Country Data Retrieval	Correct implementation of `get_all_countries` to retrieve and return a list of distinct countries in ascending order, with "All" appearing first.	2
RQ2_2	Customer Statistics Calculation	Accurate implementation of `get_statistics` to calculate and return the correct statistics for customers in the selected country.	2
RQ2_3	Display of Aggregated Statistics	Proper rendering of the aggregated statistics for the selected country at the bottom of the table.	2
RQ2_4	Error Handling for Non-existent Countries	Proper error handling and user redirection with an appropriate message when a non-existent country is requested.	2

Table 3: Marking Criteria for Task 2.

2.3 Task 3: Generating Invoices (12 Marks)

2.3.1 Task Brief

An invoice is a record/document that provides a detailed summary of a customer's purchases. Each customer may have multiple invoices reflecting various transactions over time. In the iMusic system, only complete albums can be purchased; individual tracks are not available. An album consists of several tracks, and customers can buy one or more albums in a single transaction. A customer can only buy a particular album once in a single transaction.

Your task is to implement functionality for generating invoices for customers. When a user navigates to the `/invoice/` route, the page should display a form with a dropdown menu listing all customers retrieved from the `Customer` table in the database. The customers' names should be displayed in the format `Firstname LASTNAME`, where `Firstname` is the customer's first name and `LASTNAME` is their last name in uppercase. There should be a single space between the first name and last name. The associated `value` for each customer in the dropdown should be their `CustomerId`, and customers should be sorted in ascending order by name (`Firstname LASTNAME`). The function `get_all_customers` should be completed to retrieve the customer data from the database. The `get_all_customers` function should be called from the `invoice` function, which is triggered when a user navigates to the `/invoice/` route, with the customer data passed to the template for rendering.

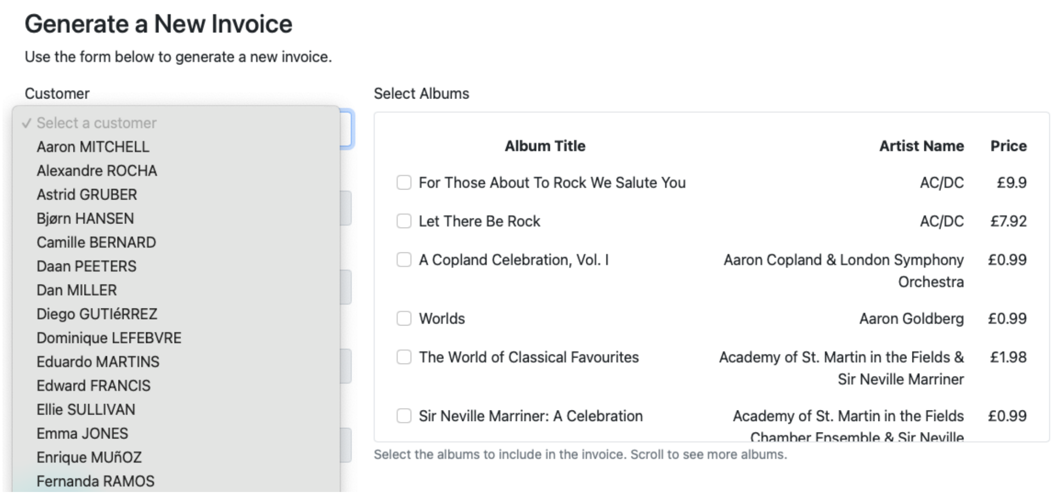
Upon selecting a customer from the dropdown, the form fields for `Address`, `City`, `Country`, and `Postal Code` should automatically populate with the customer's details from the database. This interactive feature is already implemented; your focus will be on ensuring that the correct customer details are retrieved and linked to the appropriate template fields during the rendering process. You do not need to allow users to edit customer details or manually set invoice information. However, it is important to note that the `InvoiceDate` field should be set to the current date using the SQL function `DATETIME('now')` - you should use this directly in the SQL query to set the `InvoiceDate` field in the `Invoice` table. Any missing details (such as `State`) from the form do not need to be addressed for this task.

The form also includes a table-like component displaying the `Album`s available for purchase (all albums in the database). This interface allows users to select multiple `Album`s to include in the invoice. The table should present the following information:

- The **Album Title** and **Artist Name** fields, displayed as text.
- The **Price**, which is the sum of the prices of all tracks in the album, rounded to two decimal places.

Albums should be listed in ascending order, first by the artist's name and then by the album title. The user should be able to select multiple albums to include in the invoice. The album information should be retrieved from the `Album` and `Track` tables in the database. The function `get_all_albums` should be completed to retrieve the album data from the database.

The functions `get_all_customers` and `get_all_albums` called from the `invoice` function should which is triggered when a user navigates to the `/invoice/` route.



Generate a New Invoice

Use the form below to generate a new invoice.

Customer

✓ Select a customer
- Aaron MITCHELL
- Alexandre ROCHA
- Astrid GRUBER
- Bjørn HANSEN
- Camille BERNARD
- Daan PEETERS
- Dan MILLER
- Diego GUTIéRREZ
- Dominique LEFEBVRE
- Eduardo MARTINS
- Edward FRANCIS
- Ellie SULLIVAN
- Emma JONES
- Enrique MUñOZ
- Fernanda RAMOS

Select Albums

Album Title	Artist Name	Price
☐ For Those About To Rock We Salute You	AC/DC	£9.9
☐ Let There Be Rock	AC/DC	£7.92
☐ A Copland Celebration, Vol. I	Aaron Copland & London Symphony Orchestra	£0.99
☐ Worlds	Aaron Goldberg	£0.99
☐ The World of Classical Favourites	Academy of St. Martin in the Fields & Sir Neville Marriner	£1.98
☐ Sir Neville Marriner: A Celebration	Academy of St. Martin in the Fields Chamber Ensemble & Sir Neville	£0.99

Select the albums to include in the invoice. Scroll to see more albums.

Figure 4: An example of the invoice page. Please note that the actual data displayed may differ, and this is for illustrative purposes only.

When the user presses the "Submit" button, the form data will be `POST`ed to the `/generate_invoice/` route, triggering the `generate_invoice` function. This function should validate the submitted data before calling the `process_invoice_in_db` function to create the invoice.

The `process_invoice_in_db` function should handle the database logic to create a new invoice and should only be called when valid data is provided. It must use the existing function parameters to complete the task. The function should create a new invoice for the selected customer, ensuring it is linked to both the customer and the tracks associated with the chosen albums. Before completing the `process_invoice_in_db` function, review the database schema and the existing invoices to understand how the invoice data is structured.

If a user provides data in their submission that does not exist in the database, the system should redirect them back to the `/invoice/` route and provide an appropriate error message. The error message should be displayed in the same format as the error messages in the provided template file, using the `flash` function in Flask. Refer to Table 4 for a complete list of error messages applicable to different scenarios.

Alert/Error Message	Alert Type	Description
Invalid customer selected	danger	The selected customer does not exist in the database.
Invalid album selected	danger	One or more selected albums do not exist in the database.
An error occured	danger	Any other error that occurs during the invoice generation.
Invoice generated successfully	success	The invoice was successfully generated and stored in the database.

Table 4: A list of error messages that should be displayed for different scenarios in Task 3.

2.3.2 Evaluation Procedure

When testing your solution, the iMusic team will:

1. Navigate to the `/invoice/` route on the website.
2. Select a customer from the dropdown menu.
3. Verify that the form fields for `Address`, `City`, `Country`, and `Postal Code` are automatically populated with the customer's details.
4. Select one or more albums from the table.
5. Submit the form to trigger the `generate_invoice` function.
6. Verify that the invoice is correctly generated and stored in the database.
7. Verify that the invoice details are correctly linked to the selected customer.

To further test your solution, the iMusic team will attempt to generate invoices for invalid customers and invalid album selections. Ensure that your solution can handle any valid data in the database, not just the examples provided. You are encouraged to test your solution with different data to ensure it works correctly—you will need to create your own test data for this task. The iMusic team will not provide you with additional test data.

2.3.3 Constraints

Your implementation must meet the following requirements:

- The entire implementation must be contained within the `invoice`, `get_all_customers`, `get_all_albums`, `generate_invoice`, and `process_invoice_in_db` functions. Do not create additional functions or modify the template file.
- The implementation must run as a Flask application, using the command `python iMusic.py` or `python3 iMusic.py`.
- Use the `sqlite3` module exclusively for database interactions. Other modules or approaches (e.g., SQLAlchemy) are not permitted.
- Do not modify the provided template file (`templates/invoice.html`).

2.3.4 Marking Scheme (12 Marks)

Task 3 is worth 12 of the 25 marks available for the assignment. The following criteria will be used to assess your work:

ID	Requirement	Details	Marks
RQ3_1	Customer Selection	Correct implementation of customer dropdown list, displaying customer names in `Firstname LASTNAME` format and sorted by name in ascending order.	2
RQ3_2	Customer Details	Accurate retrieval and display of customer address, city, country, and postal code fields upon customer selection.	2
RQ3_3	Album Display	Correct display of available albums information, sorted by artist and album title.	2
RQ3_4	Invoice Creation	Proper implementation of the `generate_invoice` and `process_invoice_in_db` functions, creating a new invoice with the correct billing data and linking it to the selected customer.	4
RQ3_5	Error and Success Handling	Proper error handling and user redirection with appropriate messages.	2

Table 5: Marking Criteria for Task 3.

3 Submission

To complete the assignment, submit an updated `iMusic.py` file containing your solutions to the tasks. No other files should be included. Ensure that your code is well-commented, and include your name and student ID at the top of the file. Be sure to modify only the functions specified in the assignment brief. Avoid adding additional functions or modifying the provided template files. Submit your solution via Moodle by the deadline - 19 December 2024 at 15:00.

3.1 Penalties

- **Late Submission**: Standard University policy - 5% penalty per working day late.
- **Incorrect File Name**: Submitting a file with an incorrect name will result in a 10% deduction from your final mark.
- **Use of External Libraries**: Only the `sqlite3`, `csv`, `flask` and the python standard library modules are permitted. Use of any other libraries will result in no marks awarded for that functionality. This may also cause runtime errors during testing by the iMusic team, leading to a significant mark deduction.

3.2 Academic Integrity

By submitting your work for assessment, you confirm that it is your own. Please familiarise yourself with the Academic Misconduct policy. Remember:

- **Code Sharing**: Do not share your code or specific approach to solving the task with others, particularly in shared living spaces.
- **Code References**: Reference any external code or resources used, and ensure you understand and can explain any referenced material.
- **Protect Your Work**: Keep your work secure and regularly backed up, and do not allow others access to your files or computer. Be mindful of security in shared spaces.
- **Purpose of Assignment**: This assignment is designed to assess your understanding and application of course material. If you have questions about academic misconduct, please contact the module convenor.

Remember, your goal is to learn - not simply to get a grade. If you need support, make use of lab sessions and office hours.

Acknowledgements

The data used in this assignment is based on the Chinook Database [1]. Please note that the data provided in the database was not generated by the module team and may not be accurate or complete. The iMusic scenario is entirely fictional and does not represent a real company. We'd like to thank Jane Zhao, Huimin Tang, and Yue Yang for their help reviewing this assignment.

[1] Chinook Database: https://github.com/lerocha/chinook-database