Design and Development of a Customer Management System in the Food Industry

Mariya Zhekova[1], *Nedyalko* Katrandzhiev[1*] and *Mario* Petkov[1]

[1]University of Food Technology, 26 Maritza Blvd, Plovdiv 4002, Bulgaria

Abstract: The article presents the design and customer management system development process using Python, PostgreSQL, and Tkinter in the food industry. The application aims to provide the ability to perform all CRUD operations related to business processes and make dynamic inquiries related to customer management. It stores information about customers, materials and suppliers, generates reports and provides business process management. The system is designed as a desktop application for small and medium business owners who can make complex management decisions based on the information received. The application is suitable for laboratories, hotels, stores selling food and household goods, various types of orders and services related to food products etc. For its creation, a programming language was used – Python, a relational database – PostgreSQL, for the GUI implementation– Tkinter and for version control – Git and GitHub. In the implementation we have adhered to all good programming practices. The detailed description of the program code aims to demonstrate its efficiency, functionality, simplicity, flexibility and adaptability. Creating the database and the relationships between the objects demonstrate the program integrity, security and normalisation of the data. The application implementation is a step towards the food industry's digitalisation making it more flexible and adaptable to market changes. The system helps small and medium-sized business owners make important management decisions that determine their success and prosperity. It is suitable for small startup projects in most cases do not have a large budget for purchasing customer management software.

1 Introduction

In the modern world, digitisation and automation of business processes are becoming increasingly important for the success and prosperity of many organisations. Customer management and process optimisation are actions of key importance and impact on the success and prosperity of any business. The article presents the design and development process of creating a customer management system in the food industry using Python, PostgreSQL and Tkinter. The main objective of the system is to facilitate the management of customers, materials and suppliers, the generation of inquiries and reports, providing an easy-to-use and efficient platform for businesses such as hotels, convenience stores, grocery stores, laboratories, etc. Using the Python language allows for fast and flexible development and PostgreSQL provides database reliability and scalability. The Tkinter-based graphic interface makes the system intuitive and accessible to users.

2 Related Work

Customers are the most important part in information systems [1]. Key factors in this collaboration are social interactions, customer communication and effective management of business processes such as resource planning, supply chain management, etc., united in a common effective and sustainable data management system. Customer-manager relationships in a customer management system (CMS) are organised in such a way as to increase customer satisfaction and the effectiveness of managerial marketing efforts [2]. Agile methodologies focused on customer satisfaction have been adopted in many software projects [3]. According to [3], the trust relationship between project team members and customers is a key identifier in Agile.

* Corresponding author: n_katrandzhiev@uft-plovdiv.bg

Research topics in business services can be divided into two groups – examining macro problems and economic contributions and examining processes in a company at the micro level [4]. Some researchers examine the closeness of supplier-customer relationships. Others analyse the competitive and marketing implications of customer service organisations. Still others seek to ensure the confidentiality of customer information. The authors in [5] emphasise the importance of trust between the service provider and customers.

With increasing competition and the need to improve customer service, companies in the food industry are looking for ways to automate and optimise customer management and the booking of goods and services. Various software solutions for customer and reservation management, customer communication, product tracking, and marketing strategies exists.

One example in this area is the Planfy platform which provides a complete customer management solution. It offers the ability to connect to social networks, thus improving the visibility of business services. With Planfy, business web pages can also be created that present services, prices, teams, products, working hours, etc. Planfy also offers an online calendar for managing employees by planning and synchronising their work schedules [6]. The system sends automated reminders for reserved products and services, reducing the risk of missed deliveries, reservations, and purchases and encouraging customers to book again. The built-in database (DB) automatically stores information about customers during each visit, making it easier to stay in touch with them and plan future marketing campaigns. The platform reaches more potential customers through marketing strategies and supports various payment methods [6].

Timify is another competitive solution for business users and management, suitable for various businesses. It optimises workflows and resource management by integrating and automating administrative tasks. The platform offers the ability to add QR codes to business cards and invoices. Timify automates many administrative tasks such as work schedules, customer reservations, rescheduling and cancelling visits, payments and invoicing [7]. It offers integration with Google Reserve, which allows customers to make reservations directly from the Google Search results page or Google Maps. It allows management of various resources – from staff to equipment and premises. It can also manage individual and group services and create individual offers. The platform provides a calendar for staff, allowing them to add schedules and manage reservations and customer visits [7].

In the food industry, search applications have been created with strictly fixed criteria for selecting catering establishments, which allow multi-factor filtering and evaluation of establishments in terms of the interests of different user groups [8]. Another type of application is for online food ordering. They adapt to the customer needs and can dynamically manage their orders in real-time [9]. Most CMS are created to track and predict customer satisfaction, i.e. the focus is on the customer [10], unlike the application proposed in this study, which is designed to benefit managers. Other applications focus on improving management and exercising control over processes by managers [11].

Although many solutions are developing a new software solution for customer and service management is an opportunity to create a flexible and effective system that is easy to use and adapts to different businesses. The development of systems for customer management in the food industry is a hot topic in the constantly changing market needs. Customer management is a key part of the supply chain in the food industry, where, together with the established sales network, home deliveries, inventory management, distribution network and service build a strategy for intelligent development and a guarantee of success for each manufacturer.

The applied software application is free, lightweight, and extremely flexible and offers easy and fast service to current and new customers, without engaging the user's attention with advertisements. It is built with a wide user interface, characterised by few levels and many options, and is powered by the three-click rule – users can access any information or functionality within three or fewer clicks. The software is designed to work on a single computer. If necessary, it can be used remotely in a local network and or through the global network using VPN. This ensure secure communication.

The other software reviewed is characterised by a deep interface with many levels, which makes it difficult for the user to navigate. For their full functioning, a fee is often required. The limited free versions constantly bother the user with advertisements and prompts to purchase a license. Some of them are cloud-based with a paid monthly license and all customer information is stored on a remote server, not with the user. Sometimes this also causes information to leak outside the company using the software. At the same time, the user becomes dependent on the company that created the software, since the entire database with customers and information about them is located on the server of the company offering the software.

3 Tools and Resources Used to Create the Application

Python was chosen to develop the application because the language is clear and intuitive and has many built-in and easy-to-use features [12]. Python supports object-oriented programming (OOP). In the context of a customer management system, OOP allows the creation of separate classes for customers, reservations, goods

and materials and services, which facilitates data management for the various functionalities.

The **PyCharm** development environment was used to develop the application, which offers full support for Python including syntax highlighting, automatic code completion, real-time error checking and refactoring [13]. Creating a Python project with a virtual environment is a basic practice in software development. The virtual environment provides an isolated environment for each project so that the project's dependencies are not confused with the dependencies of other projects or with packages installed on the system. **DataGrip** is the environment used to create and manage databases. It is designed to work with various database management systems and provides convenient tools for managing, editing and testing databases.

PostgreSQL is a Relational Database Management System RDBMS. It stores information in tables that interact with each other through relationships. PostgreSQL provides primary and secondary keys to link between tables. It also supports the ACID properties (Atomicity, Consistency, Isolation and Durability). **SQL** (Structured Query Language) allows users to perform CRUD (create, read, modify, delete) operations on tables. Data in a database is stored in tables representing related data. Attributes of a certain data type (fields) are recorded in the columns and records (values) are recorded in the rows. The project also uses the **Git** version control system to track code changes, create branches and work as a team on the same project. **GitHub** is an online platform hosting Git repositories, making it easy for developers to share and collaborate.

Python libraries **tkinter, tkcalendar** and **datetime** was used to build the graphical user interface (GUI). The **tkinter** library provides an interface for creating graphical applications and does not need to be installed separately. It is powerful and flexible and supports various types of graphical components that can be used to make windows, buttons, menus, text boxes, etc. [14-16]. **Themed Tkinter Widgets (ttk)** is a tkinter submodule that provides additional, more modern and themed graphical components. In ttk, there are variants of the standard tkinter components with improved appearance and behaviour – buttons, labels, tables, etc. [14]. **Messagebox** is a part of tkinter library that works with dialogue boxes to display warnings, errors, queries and informational messages. This adds interactivity in communication with users [14]. **Tkcalendar** provides easy-to-integrate calendar components (DateEntry and Calendar) built with tkinter. This allows users to easily and intuitively select dates through a visual calendar interface [15]. The DateEntry component is a text field with a built-in drop-down menu for selecting dates [16]. **Datetime** library [15] provides classes for working with dates and times. It is useful for manipulating time data, calculating differences between dates, formatting dates, etc. The datetime library is used in various applications, including this customer management project, handling and displaying the dates of visits, purchases and deliveries, correctly reflecting periods in reports.

4 Designing the Database and the Structure of the Customer Management System

4.1. System architecture

4.1.1. File and directory structure

MarioPetkov-ManagementSystem/ is the main directory of the project and it contains the virtual environment and the program code to implement the software.

backend/ – subdirectory where the scripts are divided into segments.

- appointment_view.py – Reservations tab
- customer_history.py – Customer Reports tab
- customer_view.py – Customers tab
- database.py – module for connecting and interacting with the database
- employee_history.py – Employee Reports tab
- employee_view.py – Employees tab module
- history_view.py – Reports tab module
- main.py – start and manage the application
- material_view.py – Materials tab module
- person.py – module with static methods used by other modules
- service_view.py – Services tab module

dist/ – directory that contains installation files

build/ – directory with all temporary files

source/ – directory with the project's source code

- get-pip.py – pip install script that is used only once when creating the project
- README.md – project documentation
- requirements.txt – project dependencies file
- ManagementSystem.spec – packages the entire project into a single file and runs even without Python on the OS.

4.1.2. Description of the main modules

The system operates under a unique algorithm created by the authors, which offers fast, reliable and flexible customer service. The algorithm is not the subject of this publication. Only the modules that participate in it and what they are used for are described here. The algorithm itself is to be considered in a separate publication.

The authors have an idea to integrate an AI assistant into the software, which will learn from the user's actions and will offer automated operations.

Table 1. Main modules description

File	Description
main.py	contains the basic logic for running the app. It connects to the DB after calling the Database() class. The main window is initialised and all tabs defined
database.py	contains all functions and methods related to interaction with the PostgreSQL DB, including connection logic, preparing SQL queries and processing the results
person.py	contains methods and helper functions that are used by other modules. Include input validation, format conversion etc.
*_view.py	initialise a specific tab in the GUI
employee_view.py	contains the logic to check past reservations with the status Saved
history_view	contains a table with all existing reservations, has a search engine that can filter by any column and produces a report of visits and statistical general information for the last 30 and 7 days
customer_history.py	creates a Reports for reservations made and funds spent. Summarises data
employee_history.py	creates a Reports for each employee's income and expense, summarise information for the last 30 and 7 days

4.2. Application Database Design

Entity Relationship (ER) Diagram shows how the entities in the DB are related (Fig. 1).

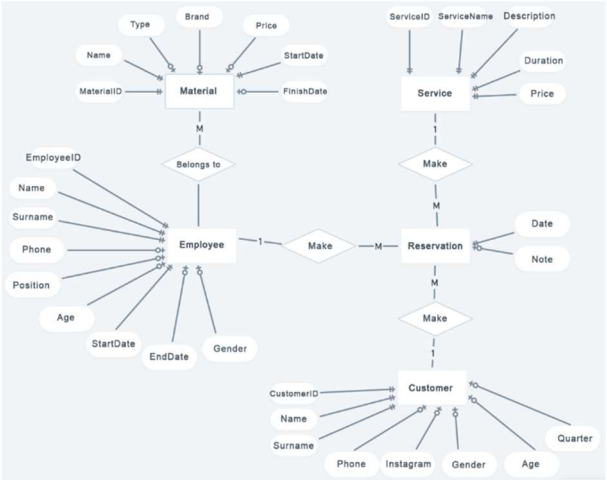

Fig. 1. Entity Relationship Diagram

Schema visualisation is a handy tool in database design and maintenance. The scheme is intuitive and presents the relationships between objects (Fig. 2).

The software can also be used in a local network or WAN via VPN by running it on a remote computer. Successful launch of the software requires an active user and password.

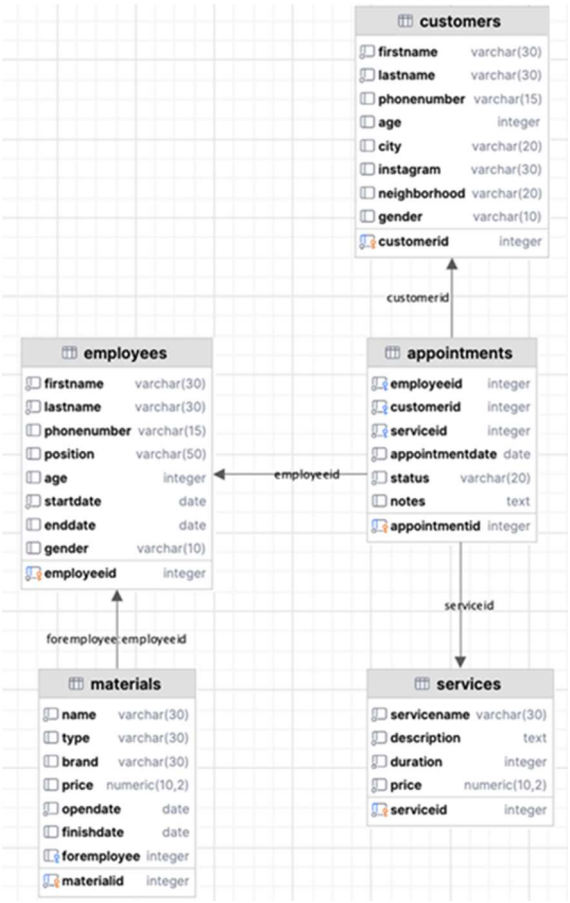

Fig. 2. Visualisation database schema

4.3. Building a Graphical User Interface

The Employees module consists of three Label Frames:

- Add new employee
- Edit employee
- Search

There are also three buttons Add, Edit and Delete. In the Add and Edit sections are fields for filling in input data: *Name, Surname, Phone, Age, Position, Start day, Last day* and *Gender* – with mandatory fields marked with "*" (Fig. 3).

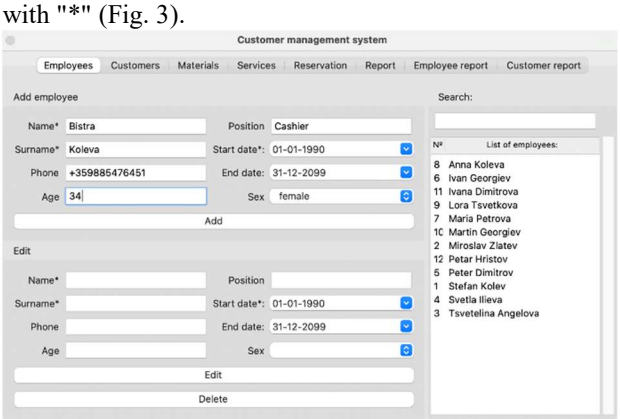

Fig. 3. Employee module view

The Search tab contains a list of all employees, sorted alphabetically, that have already been added to the database. The search can be carried out by part of a name or surname.

When editing an employee, his name is first marked in the list and the data is loaded into the Employee update tab. The Edit button saves the changes and a message appears on the screen: The employee has been updated! On failure, a window is displayed with a description of the error. **When deleting** an employee, his name is searched and marked in the list of employees and his data is loaded in the Edit Employee window. The Delete button deletes the entry. A message appears on the screen: The employee has been deleted!

There are three subsections in the **Reservations menu** (Fig. 4):

- Add reservation
- Edit
- Search

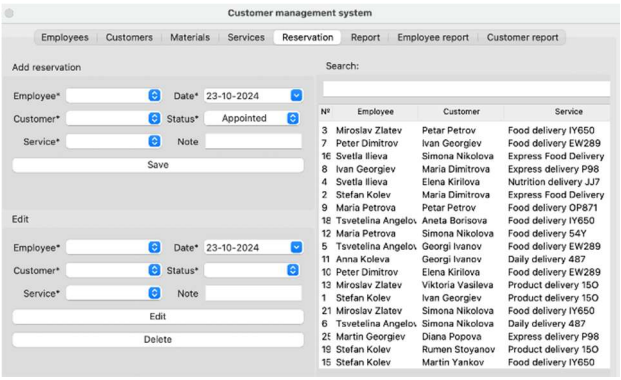

Fig. 4. Reservation module

The Reports module contains a Search subsection and three read-only fields. The table shows existing reservations arranged chronologically in descending order. The search field also allows you to search by part of a name in each column. The fields at the bottom of the window provide summary information on the number of reservations for the last 30 and 7 days and the entire reference period (Fig. 5).

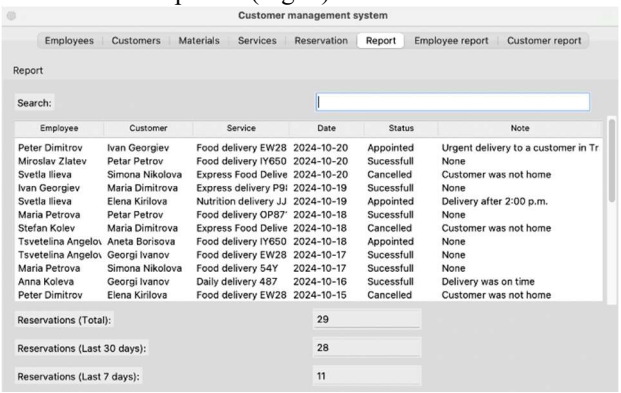

Fig. 5. Reports module

When a filter is set in the Search field, reservations that meet the condition are displayed. The report dynamically calculates how many reservations have been made for the last 30 and the last 7 days.

This section presents user interfaces (not all) for performing basic operations related to business processes and dynamic queries on data. The system has been tested with experimental data, which are visible in the presented figures. The test results are successful.

5 Implementation of the customer management system

5.1. Creating a Local PostgreSQL Database

- Install PostgreSQL via Ubuntu/Debian terminal
- Starting a server
- Connect with Postgre server with Postgre account
- Create a new user and a new database
- Granting rights to the user on the database

5.2. Create the Application Database

Employee table

```
CREATE TABLE Employees (
        EmployeeID SERIAL PRIMARY KEY,
        FirstName VARCHAR(30) NOT NULL,
        LastName VARCHAR(30) NOT NULL,
        PhoneNumber VARCHAR(10) NULL,
        Position VARCHAR(20) NULL,
        Age INTEGER NULL,
        StartDate DATE NOT NULL DEFAULT CURRENT_DATE,
        EndDate DATE NULL,
        Gender VARCHAR(10) NULL,
        CONSTRAINT check_phone_number_employees
        CHECK (PhoneNumber ~ '^[0-9]{10}$')
    );
```

Similarly, the tables were created by Customers, Services, Materials and Reservations. Cascading delete is used in the Materials table. Material can be deleted when it belongs to an employee who has been deleted. A reservation can only be deleted completely when the customer, employee or service it is associated with is deleted.

5.3. Programmatic Implementation of the Main Module main.py

The **main.py** module is the main application file. It initialises the main GUI window and creates the modules that manage the functionalities of the customer and service management system. Firstly, the necessary libraries, modules and classes from other files are imported into the project. The **MainApp** class is the main application class that creates the main window and loads the various tabs. The constructor of the MainApp **__init__(self, root)** class initialises the main

elements of the main window and the database connection is created by an object from the Database class. Creates a main frame that is added to the main window (root). Creates a Notebook widget that will contain all tabs. Calls the center() and setup_views() methods. The **center(self)** method centres the main window on the user's screen. Defines the screen and window size. Calculates the coordinates for centring the window on the screen. Method **setup_views(self)** create and add all modules in the Notebook widget. The **run(self)** method starts the main application loop, allowing users to interact with the GUI.

5.4. Programmatic Implementation of the Module person.py

The **person.py** module contains the Person class and several static methods to handle data related to person records, such as entering, validating and clearing fields in the GUI. Some basic data are also defined in this class: gender, quarter and customer status. The padding parameter contains values for spacing between fields and is used to unify the view in the GUI. The **get_person_data(entry_fields, option_menus)** method collects input data and returns a list of values. For a DateEntry field, the method retrieves the value in date format. For others, it returns the typed input or None if the field is empty. Method **clear_input_fields(entry_fields, option_menus)** clears all fields, returning them to their initial states. Method **check_provided_names(first_name, last_name)** checks the required First Name and Last Name fields. Method **get_valid_string(value)** checks the type of the variables whether they are of type string. Method **get_valid_int(value)** checks if the supplied value is numeric. Method **get_valid_date(value)** returns the value if it is a date or a string.

5.5. Programmatic Implementation of the Module database.py

The **database.py** file is the main module that manages the database connection and contains the basic methods for retrieving, inserting, updating and deleting data. It uses the psycopg2 library to connect to a PostgreSQL database and includes error handling to ensure stable and reliable operation. Imported libraries: **datetime** – used to work with dates and times; **messagebox** – used to display error messages for DB connection problems; and **psycopg2** – for working with PostgreSQL in Python.

The **get_time_periods()** function stores in variables the duration of the last 30 and last 7 days, starting from today's.

The Database class manages the DB connection and provides methods for executing SQL queries. The **__init__(self)** class constructor initialises the DB connection and calls the **self_connect_to_db()** connect method. The **self_connect_to_db(self)** method attempts to connect to the DB using psycopg2.connect. Method **close_connection(self)** closes the cursor and the database connection if they are initialised. Method **_execute_commit(self, sql, params=())** executes a SQL query with parameters and writes the changes to the database. In case of an error an error message is displayed (Fig. 6).

```
def _execute_commit(self, sql, params=()):
    try:
        self.cursor.execute(sql, params)
        self.connection.commit()
        print("Query completed successfully!")
    except psycopg2.Error as e:
        self.connection.rollback()
        messagebox.showerror("Error",f"An error occurred: {e}")
        self._close_connection()
    except Exception as e:
        self.connection.rollback()
        messagebox.showerror("Error",f"An error occurred: {e}")
        self._close_connection()
```

Fig. 6. Request execution

Method **_fetch_single(self, sql, params=())** selects a single record from the DB based on a SQL query and parameters. Method **_fetch_all(self, sql, params=())** extracts all records from DB based on SQL query and parameters.

5.6. Programmatic Implementation of the Employee Module

The **__init__(self, notebook, db)** constructor initialises an EmployeeView object, creates basic graphical components (tab and frame) and loads the employee interface. Method **widgets_add_employee(self)** makes and sets controls for adding a new employee, such as text fields for entering first name, last name, phone, job title, etc. Method **widgets_edit_employee(self)** makes and sets controls to edit an already existing employee, to update or delete. Method **widgets_employees_list(self)** makes and sets a list of employees and allows searching and selecting an employee for editing. Method **fill_employee_entries(self, event)** fills the edit fields with information about the selected employee from the list. Method **populate_existing_employees(self)** fill a list of all available employees from the database. Method **add_employee_clicked(self)** handles an event when a button is pressed to add an employee, collecting the data from the fields, validating them and saving them to the database (Fig. 7).

```
def add_employee_clicked(self):
    entry_fields = [
        self.entry_f_name, self.entry_l_name, self.entry_phone_num,
        self.entry_age, self.entry_position, self.entry_start_date, self.entry_end_date
    ]
    option_menus = [self.__add_selected_gender]
    values = Person.get_person_data(entry_fields, option_menus)

    try:
        Person.check_provided_names(self.entry_f_name, self.entry_l_name)
        # Person.valid_age(self.entry_age)
        # Person.valid_dates(self.entry_start_date, self.entry_end_date)
        self.db.save_employee(values)
        Person.clear_input_fields(entry_fields, option_menus)
        self.populate_existing_employees()
        messagebox.showinfo("Add customer", "Customer added successfully!")
    except Exception as e:
        messagebox.showerror("Error", f"{e}")
```

Fig. 7. Fragment for a method add_employee_clicked()

The **edit_employee_clicked(self)** method handles the Update Employee event by updating the selected employee's data in the DB. Method **delete_employee_clicked(self)** handles the Delete Employee event by removing the employee from the DB. Method **search_employees(self, event)** handles a search event for employees when text is entered in the search field, filtering the list of employees based on the entered value. Method **check_past_scheduled_appointments(self)** checks past scheduled reservations and provides an option to change their status. Method **ask_status_change(self, customer_name, appointment_date)** creates a popup that asks the user to change the status of an expired reservation. Method **set_status_and_close(self, window, status_var, status)** sets the new reservation status and closes the popup window. Method **get_existing_employees(self)** retrieves all existing employees from the database, returning their *EmployeeID, FirstName* and *LastName*. The results are sorted by FirstName and returned as a list of dictionaries (Fig. 8).

```
# Employee methods
def get_existing_employees(self):
    sql = "SELECT EmployeeID, FirstName, LastName FROM Employees ORDER BY FirstName"
    rows = self._fetch_all(sql)
    return [{'EmployeeID': row[0], 'FirstName': row[1], 'LastName': row[2]} for row in rows]
```

Fig. 8. A method of the Database class that retrieves the existing employees

Method **get_employee_info(self, employee_id)** gets detailed information about a particular employee based on the provided employee_id. Returns data such as *FirstName, LastName, PhoneNumber, Age, Position, StartDate, EndDate* and *Gender* in a dictionary. Method **update_employee(self, updated_data, employee_id)** updates the data of an existing employee in the DB using the provided new values and employee_id. Method **save_employee(self, values)** saves a new employee to the DB with the provided values. If any values are empty, they are written as None in the DB. Method **delete_employee(self, employee_id)** deletes an employee from the DB based on the provided employee_id. Method **search_employees(self, search_term)** searches for employees in the database using the provided search_term keyword.

Similarly, the modules Customers, Materials, Reservations and Reports for employees and customers were created. Their functionality is made according to the specific requirements of the module.

5.7. Programmatic Implementation of the Customer Module

Table 2. Additional functions in class Customers

Method	Description
widgets_add_customer(self)	Creates and arranges all graphical components to add a new customer to the visual interface.
widgets_edit_customer(self)	Creates and arranges graphical components for editing existing customer information, including input fields and buttons to update or delete the customer.
widgets_customers_list(self)	Initialises and arranges a table and a lookup field to display and search customers and select to edit or delete.
populate_existing_customers(self)	Loads and displays all existing customers from the database in the customer treeview.
fill_customer_entries(self, event)	Automatically fills in customer info edit fields when double-clicking a customer in the list by retrieving the data from the DB.
add_customer_clicked(self)	Collect user input for a new customer and save it to the DB while updating the customer list and displaying a success or error message.
edit_customer_clicked(self)	Collect the updated data from the user fields and update the information in the DB, then refresh the customer list.
delete_customer_clicked(self)	Deletes a selected customer from the DB and refreshes the customer list while displaying a success message.
search_customers (self, event)	Searches for customers based on the search entered by the user, updating the list of results that match the search.
get_existing_customers(self)	Retrieves and returns a list of all existing customers from the DB, sorted by name.
get_customer_info(self, customer_id)	Retrieves and returns detailed information about a specific customer from the DB by the customer ID.
update_customer(self, updated_data, customer_id)	Updates a customer's data in the database with the new submitted values for the corresponding customer ID.
save_customer(self, values):	Writes a new customer to the DB with the given values for name, surname, phone, age, city, Instagram, quarter and gender.
delete_customer(self, customer_id):	Deletes a customer from the database by the supplied customer ID.

5.8. Program Implementation of the Employee Reports Module

The constructor **__init__** of the class initialises the AppointmentsHistory class, creates an interface for the References tab and configures the search and display fields for appointments. Loads all reservations and updates statistics on app launch. Method **setup_ui** sets up the GUI, including search fields, statistics and reservation table. Adds a vertical scroll bar and sets column sizes and table titles. The **search_history** method searches for bookings based on the input data and updates the displayed results and statistics. Calls methods to search the database and update statistics. Method **update_appointment_counts** updates the fields that show the number of reservations for a period. The **display_results** method clears the data in the Reservations table, formats and displays it in a table. Method **populate_all_appointments** loads all appointments and updates data on initialisation. Call methods to display summary results and updates. Method **fetch_all_appointments_ordered** retrieves all appointments from the database, sorted by criteria and direction. Formats the results and returns them as a list of dictionaries.

Table 3. Additional functions in class AppointmentsHistory

Method	Description
populate_employees	Loads a list of all employees from the database, sorts them by name and updates the dropdown with the employee names.
on_employee_selected	Processes the selection of an employee from the drop-down menu, updates the employee's information and loads the employee's assignment and earnings data.
populate_employee_appointments	Checks to see if an employee is selected and loads and displays that employee's assignments in the table.
populate_employee_earnings	Loads and updates the selected employee's earnings information by updating the total earnings, last 30 and last 7 days.
populate_money_spent	Updates the employee expense fields to include total expenses, last 30 days and 7 days expenses.
display_results	Deletes the current data in the table and adds the new results obtained from the DB. Visualises info about clients, services, reservation dates and notes in the table.
get_employee_appointments	Executes the SQL query to retrieve all assignments of the specific employee from the DB, incl. customer and service info.
get_employee_earnings	Runs a SQL query to get the total earnings and the last 30 and 7 days' earnings for the particular employee.
get_money_invested	Runs a SQL query to calculate the employee's total the expenses for the last 30 and 7 days.

Similarly, according to their logic, the Customer References were made.

Future improvements to the project would lead to increased database security. Moving from a local DB to a hosting provider would improve support, data security and consumption of non-local machines. Creating authentication functionality and user hierarchy would also increase security. It is possible to add functionality to print the schedule and reports on paper, etc.

6 Conclusion

The article presents the design and customer management system development process using Python, PostgreSQL, and Tkinter in the food industry.

The application aims to provide the ability to perform all CRUD operations related to business processes and make dynamic inquiries related to customer management. It stores information about customers, materials and suppliers, generates reports and provides management of the business process. The system is designed as a desktop application for small and medium business owners who can make complex management decisions based on the information received.

The application is suitable for laboratories, hotels, stores selling food and household goods, various types of orders and services related to food products etc.

For its creation, a programming language was used – Python, a relational database – PostgreSQL, for the GUI implementation– Tkinter and for version control – Git and GitHub.

In the project implementation, we have adhered to all good programming practices. The detailed description of the program code aims to demonstrate its efficiency, functionality, simplicity, flexibility and adaptability. The process of creating the database and the relationships between the objects demonstrate the program integrity, security and normalisation of the data.

The application implementation is a step towards the food industry's digitalisation making it more flexible and adaptable to market changes.

The system helps small and medium-sized business owners make important management decisions that determine their success and prosperity. It is suitable for small startup projects, which in most cases do not have a large budget for purchasing customer management software.

The system has shown interest and has been implemented, which proves its usefulness. The data in the photos attached to the publication are experimental, for reasons of personal data security.

References

1. M. Deeg, Privilege Escalation via Client Management Software. SySS GmbH (2015)
2. L. Cricelli, FM. Famulari, M. Greco, M. Grimaldi, Searching for the one: Customer relationship management software selection. J. Multi-Crit Decis Anal., **27:173–188** (2020) https://doi.org/10.1002/mcda.1687
3. S. Vaezitehrani, Customer knowledge management in global software projects. Master Thesis. Chalmers University of Technology Northumbria University, Gothenburg, Sweden 2013
4. L. Bagdoniene, A. Kazakeviciute, The Model of Client Relationship Management of a Knowledge Intensive Business Services Organization. Social Sciences ISSN:1392-0758, **65(3)**, (2009)
5. M. Kautonen, M. Hyypia, Knowledge intermediaries or routine service producers? Exploring finnish M-KIBS using the innovation system approach. Knowledge-Intensive Business Services Geography and Innovation, ISBN: 9781315591216, (Routledge, first edition, 2010)
6. Booking System for Business, https://www.planfy.com, last visited on 15.10.2024
7. Appointment Booking Software Built for Your Successful Business, https://www.timify.com/en/features/customer-management, last visited on 17.10.2024
8. L. Rozhdestvenskaya, O. Rogova, L. Cherednichenko, Digital Technologies in Managing Food Industry Enterprises. Advances in Economics, Business and Management Research, ISSN: 2352-5428, Atlantis Press. (2020). https://doi.org/10.2991/aebmr.k.200502.096
9. D. Ghelani, T. Hua, A Perspective Review on Online Food Shop Management System and Impacts on Business. Advances in Wireless Communications and Networks. **8**. 7-14. (2022). https://doi.org/10.11648/j.awcn.20220801.12
10. F. Mahawrah, I. Shehabat, E. Abu-Shanab, The impact of knowledge management on customer relationship management: a case from the fast food industry in Jordan. International Journal of Electronic Customer Relationship Management. **10, 138-157**, (2017). https://doi.org/10.1504/IJECRM.2016.10003108
11. A. Santos, T. Clemente, R. Melo, S. Machado, Integrated management system: methodology for maturity assessment in food industries. Benchmarking An International Journal. ahead-of-print, ISSN:1463-5771, 29(6), (2022) https://doi.org/10.1108/BIJ-05-2021-0280
12. R. T. Yarlagadda, L. Surya, Dr M. Patel, Python for Beginners. ISBN: 9798741059081 (REDSHINE Publications, 2021)
13. Sv. Nakov et al., Programming basics with Python, ISBN 978-619-00-0806-4. (Faber, 2018)
14. Tkinter Python interface to Tcl/Tk, https://docs.python.org/3/library/tkinter.html, last visited on 08.10.2024
15. datetime - Basic date and time types, https://docs.python.org/3/library/datetime.html, last visited on 09.10.2024
16. Tkcalendar documentation, https://tkcalendar.readthedocs.io/en/stable/documentation.html, last visited on 17.10.2024