LITHAN

Project Presentation

Module Name	Programming Fundamentals (Bundled)(SF)
Course Name	(SCTP) Professional Diploma in Full Stack Web Development (E-Learning)
Assignment Title	Employee Management System

Learner Name	Goh Jin Ying

Learner declaration

I certify that the work submitted for this assignment is my own and research sources are fully acknowledged.

Student signature: Goh Jin Ying Date: 13-5-2025

Content Page

1. Requirement Specification Document

Project: BitFutura Employee Management System (EMS)
Version: 1.0
Date: 19-04-2025
Author: Goh Jin Ying

Table of Contents

---

1. Introduction

1.1 Purpose

The purpose of this Functional Requirements Specification (FRS) is to detail the functional and non-functional requirements for the BitFutura Employee Management System (EMS). This document serves as a guide for developers, testers, project managers, and stakeholders (including HR personnel) to ensure the developed system meets the defined needs for managing employee data efficiently and supporting organizational growth at BitFutura.

1.2 Scope

This document covers the functional requirements for the BitFutura Employee Management System. The system will provide capabilities for:

- **In Scope:**
  - Adding new employee records (ID, name, department, salary, contact details).
  - Viewing the details of a specific employee.
  - Updating existing employee records.
  - Deleting employee records.
  - Listing all employees currently in the system.
  - Generating reports summarizing employee data grouped by department.
  - Sending an automated confirmation email upon successful addition of a new employee.
  - Basic user authentication and authorization (implied for data security).
- **Out of Scope:**
  - Payroll processing and generation.
  - Employee performance management or reviews.
  - Leave/attendance tracking.
  - Advanced data analytics or predictive reporting beyond the specified department report.
  - Integration with external HR platforms, accounting software, or directory services (other than the email system for notifications).
  - Self-service portal for employees.

1.3 Definitions, Acronyms, and Abbreviations

- **CRUD:** Create, Read, Update, Delete. Standard data operations.
- **EMS:** Employee Management System. The system being specified.
- **FRS:** Functional Requirements Specification. This document.
- **HR:** Human Resources. The primary user department.
- **ID:** Identifier. A unique code assigned to each employee.
- **NFR:** Non-Functional Requirement. Specifies quality attributes of the system.

- **SMTP:** Simple Mail Transfer Protocol. Used for sending emails.
- **UI:** User Interface. How users interact with the system.
- **UAT:** User Acceptance Testing. Testing by end-users.
- **UC:** Use Case. A description of user-system interaction to achieve a goal.

1.4 References

- BitFutura Assignment Brief. PDWD-PFS-FA01. Singapore: Lithan Academy.

2. Overall Description

2.1 Product Perspective

The Employee Management System (EMS) is a new, standalone application commissioned by BitFutura to manage its growing workforce data. It is intended for internal use, primarily by the HR department. The system will replace potentially manual or less efficient existing processes. It operates independently but requires interaction with an external email system (via SMTP) to send enrollment notifications. Future versions might integrate with other internal systems, but this is not within the scope of the current version.

2.2 Product Functions

The major functions of the EMS include:

- **Employee Data Management:** Providing full CRUD capabilities for employee records.
- **Employee Listing:** Offering a view of all employee records.
- **Reporting:** Generating department-wise summaries of employee data.
- **Notification:** Sending automated email confirmations for new employee enrollments.
- **User Access Control:** Ensuring secure access to the system functions and data.

2.3 User Classes and Characteristics

- **HR Administrator:**
    - **Characteristics:** Primary users of the system, familiar with HR processes and employee data requirements. Possess basic computer literacy.
    - **Needs:** Efficient data entry, accurate record keeping, ability to quickly find and manage employee information, generate reports for analysis and decision-making.
    - **Permissions:** Full access to all system functions (CRUD, List All, Reports).
- **(Optional/Assumption) Manager:**
    - **Characteristics:** Heads of departments, need access to information about their team members.
    - **Needs:** View employee details within their department, generate reports specific to their department.
    - **Permissions:** Read-only access to employees in their assigned department(s), permission to generate department-specific reports.

2.4 Operating Environment

- The EMS will be a web-based application.
- Users will access the system via standard modern web browsers, including Google Chrome, Mozilla Firefox, and Microsoft Edge (latest versions).
- The application backend will be hosted on a central server running a Linux-based operating system (e.g., Ubuntu Server).
- A relational database management system (RDBMS), such as PostgreSQL or MySQL, will be used for data persistence.
- The server must have network connectivity to allow user access and connection

to the company's designated SMTP server for email dispatch.
- An active internet connection is required for users to access the system.

2.5 Design and Implementation Constraints

- The system must manage the following employee data fields: ID (unique), name, department, salary, contact details.
- The system must provide the core functionalities: Add, View, Update, Delete, List All Employees, Department-Wise Report, and Email Confirmation on enrollment.
- The design of the Department-Wise Report layout must conform to the specifications in Appendix 5.2.1.
- All handling of personal and salary data must comply with relevant data protection regulations (e.g., GDPR, local privacy laws).
- The system must utilize BitFutura's designated SMTP server for sending emails. Configuration details for the SMTP server will be provided separately.
- (Assumption) The user interface should be intuitive and follow common web application design patterns for ease of use.
- (Assumption) The technology stack used should be modern, maintainable, and suitable for a growing technology company (e.g., Python/Django, Java/Spring, Node.js/React - specific stack TBD).

2.6 Assumptions and Dependencies

- **Assumptions:**
    - Users possess basic computer literacy and familiarity with web applications.
    - Users have access to a compatible web browser and internet connection.
    - The structure of departments within BitFutura is well-defined and available.
    - Employee IDs are unique and a format/policy for generating them exists or will be defined.
    - The content/template for the confirmation email will be provided or agreed upon.
- **Dependencies:**
    - Availability and reliability of the hosting server infrastructure.
    - Availability and reliability of the database server.
    - Availability, configuration, and reliability of the designated SMTP server for email notifications.
    - Accurate department information within BitFutura.

3. Functional Requirements

3.1 Functional Requirements

FR-01: User Authentication

- **Title:** User Authentication
- **Description:** The system shall require users to log in using a unique username and password before accessing any functionality.
- **Priority:** Must-have
- **Acceptance Criteria:**
    - Users are prompted for username and password on accessing the system.
    - Users with valid credentials gain access.
    - Users with invalid credentials are denied access and shown an error message.
    - Login attempts may be logged for security auditing.

FR-02: Add Employee Form Display

- **Title:** Add Employee Form Display
- **Description:** The system shall provide a data entry form for adding new employees.
- **Priority:** Must-have
- **Acceptance Criteria:**
    - The form includes input fields for Employee ID, Name, Department, Salary, and Contact Details.
    - Fields are clearly labelled.
    - Required fields are visually indicated (e.g., with an asterisk).

FR-03: Employee Data Input

- **Title:** Employee Data Input
- **Description:** The system shall allow the user to input data into the fields provided on the Add Employee form.
- **Priority:** Must-have
- **Acceptance Criteria:**
    - User can type text into Name, Department, Contact Details fields.
    - User can type alphanumeric characters into the ID field.
    - User can type numeric data into the Salary field.
    - (Optional) Department field might be a dropdown list populated from a predefined list of departments.

FR-04: Input Data Validation

- **Title:** Input Data Validation
- **Description:** The system shall validate the data entered by the user during employee addition and update.
- **Priority:** Must-have
- **Acceptance Criteria:**
    - All required fields (e.g., ID, Name, Department) must be filled.
    - Employee ID must be unique within the system.
    - Salary must be a valid numeric value (e.g., positive number).
    - Contact details should follow a basic format check (e.g., email contains '@', phone number contains digits).
    - Validation errors are clearly displayed to the user, preventing submission until corrected.

FR-05: Store New Employee Record

- **Title:** Store New Employee Record
- **Description:** The system shall persist the validated new employee data into the database.
- **Priority:** Must-have
- **Acceptance Criteria:**
    - Upon successful validation and submission, a new record is created in the employee database table.
    - All entered fields (ID, name, department, salary, contact) are correctly stored.
    - A confirmation message "Employee added successfully" is displayed.

FR-06: Send Enrollment Confirmation Email

- **Title:** Send Enrollment Confirmation Email

- **Description:** The system shall automatically send a confirmation email upon the successful addition of a new employee.
- **Priority:** Must-have
- **Acceptance Criteria:**
    - An email is triggered immediately after a new employee record is successfully saved.
    - The email is sent using the configured SMTP server.
    - The email content confirms the enrollment (specific content TBD).
    - The email is sent to a designated recipient (e.g., HR, the new employee's provided contact email - requires clarification).
    - Failures in email sending are logged.

FR-07: Search/Select Employee

- **Title:** Search/Select Employee
- **Description:** The system shall provide a mechanism for users to find and select a specific employee for viewing, updating, or deleting.
- **Priority:** Must-have
- **Acceptance Criteria:**
    - Users can locate an employee (e.g., via search by ID/Name, or selecting from the employee list).
    - The system clearly identifies the selected employee.
    - If a search yields no results, a "No employee found" message is displayed.

FR-08: Display Employee Details

- **Title:** Display Employee Details
- **Description:** The system shall display the stored details of a selected employee.
- **Priority:** Must-have
- **Acceptance Criteria:**
    - All stored fields (ID, Name, Department, Salary, Contact Details) for the selected employee are displayed.
    - Data is presented in a clear, read-only format.

FR-09: Update Employee Form Display

- **Title:** Update Employee Form Display
- **Description:** The system shall provide a form pre-populated with the selected employee's current data for editing.
- **Priority:** Must-have
- **Acceptance Criteria:**
    - The form includes fields for Name, Department, Salary, and Contact Details, filled with the employee's current data.
    - The Employee ID is displayed but is typically non-editable.
    - Required fields are indicated.

FR-10: Store Updated Employee Record

- **Title:** Store Updated Employee Record
- **Description:** The system shall persist the validated changes to an existing employee's record in the database.
- **Priority:** Must-have
- **Acceptance Criteria:**
    - Upon successful validation and submission of the update form, the corresponding employee record in the database is updated with the new values.

    - Only the modified fields are changed.
    - A confirmation message "Employee updated successfully" is displayed.

FR-11: Confirm Employee Deletion

- **Title:** Confirm Employee Deletion
- **Description:** The system shall prompt the user for confirmation before deleting an employee record.
- **Priority:** Must-have
- **Acceptance Criteria:**
    - When the delete action is initiated, a confirmation dialog appears (e.g., "Are you sure?").
    - The dialog clearly identifies the employee being deleted.
    - The deletion only proceeds if the user explicitly confirms.
    - The user can cancel the deletion action.

FR-12: Delete Employee Record

- **Title:** Delete Employee Record
- **Description:** The system shall remove the specified employee record from active use upon confirmed deletion.
- **Priority:** Must-have
- **Acceptance Criteria:**
    - Upon user confirmation, the employee record is either permanently deleted from the database or marked as inactive.
    - The deleted/inactive employee no longer appears in the main employee list or reports (unless specifically queried).
    - A confirmation message "Employee deleted successfully" is displayed.

FR-13: Retrieve All Employee Records

- **Title:** Retrieve All Employee Records
- **Description:** The system shall retrieve data for all active employees for display in the list view.
- **Priority:** Must-have
- **Acceptance Criteria:**
    - The system queries the database and fetches records for all employees not marked as inactive/deleted.
    - Key fields required for the list view (e.g., ID, Name, Department) are retrieved.

FR-14: Display Employee List

- **Title:** Display Employee List
- **Description:** The system shall display the retrieved list of all active employees.
- **Priority:** Must-have
- **Acceptance Criteria:**
    - Employees are displayed in a structured format (e.g., a table).
    - The list includes columns for key information like Employee ID, Name, Department.
    - The list should be easily scannable.
    - If no employees exist, a message "No employee records found" is displayed.
    - (Optional) Basic sorting (e.g., by Name or ID) is available.

FR-15: Select Department for Report

- **Title:** Select Department for Report

- **Description:** The system shall allow the user to specify which department(s) to include in the Department-Wise Report.
- **Priority:** Must-have
- **Acceptance Criteria:**
    - User is presented with an option to generate the report (e.g., via a dropdown list of departments, checkboxes, or an 'All Departments' option).
    - User can make a selection and initiate report generation.

FR-16: Generate Department Report Data

- **Title:** Generate Department Report Data
- **Description:** The system shall process employee data to generate the content for the Department-Wise Report based on user selection.
- **Priority:** Must-have
- **Acceptance Criteria:**
    - The system retrieves employee records matching the selected department(s).
    - Data is grouped by department.
    - Relevant employee details (e.g., ID, Name, Salary, Contact) are included for each employee within their department group.
    - (Optional) Summary statistics per department (e.g., headcount, total salary) might be calculated if specified in the report design.

FR-17: Format and Display/Download Department Report

- **Title:** Format and Display/Download Department Report
- **Description:** The system shall format the generated report data according to the specified layout and make it available to the user.
- **Priority:** Must-have
- **Acceptance Criteria:**
    - The report content is structured and formatted as per the design in Appendix 5.2.1.
    - The report is either displayed directly within the web interface or provided as a downloadable file (e.g., CSV or PDF).
    - **If no data is found for the criteria, a message "No data available for this report" is displayed.**

3.2 Use Cases

3.2.1 Use Case 1: Log In User

- **UC-ID:** UC-01
- **Title:** Log In User
- **Description:** Allows a registered user (e.g., HR Administrator) to gain access to the EMS.
- **Actors:** HR Administrator (or other defined user roles)
- **Preconditions:** User has valid login credentials. The system is accessible.
- **Main Flow:**
    1. User navigates to the EMS login page.
    2. User enters their username and password.
    3. User clicks the 'Login' button.
    4. System validates the credentials against stored user accounts.
    5. System grants access and redirects the user to their dashboard/main page.

- **Alternate Flows:**
    1. **A1: Invalid Credentials:** If validation fails, the system displays an "Invalid username or password" error message. Access is denied. Login attempts may be logged.
- **Postconditions:** User is successfully logged into the system and can access functions based on their role. OR User remains on the login page with an error message.

3.2.2 Use Case 2: Add New Employee

- **UC-ID:** UC-02
- **Title:** Add New Employee
- **Description:** Allows an HR Administrator to add a new employee record to the system.
- **Actors:** HR Administrator
- **Preconditions:** User is logged in with privileges to add employees.
- **Main Flow:**
    1. User navigates to the 'Add Employee' function/page.
    2. System displays a form with fields for ID, name, department, salary, and contact details.
    3. User enters the required information for the new employee.
    4. User submits the form.
    5. System validates the entered data (e.g., required fields, data types, unique ID).
    6. System saves the new employee record to the database.
    7. System displays a success confirmation message to the user.
    8. System triggers the sending of a confirmation email to a designated recipient (e.g., the new employee or HR).
- **Alternate Flows:**
    1. **A1: Validation Error:** If validation fails (e.g., missing required field, non-unique ID, invalid data format), the system displays specific error messages next to the relevant fields and does not save the record. The form remains populated with the user's input.
    2. **A2: Email Sending Failure:** If the record saves successfully but the email fails to send, the system should ideally log the error but still confirm the employee addition to the user, possibly with a warning about the email failure.
- **Postconditions:** A new employee record is successfully created and stored in the database, and a confirmation email is sent. OR The employee record is not created, and error messages are displayed.

3.2.3 Use Case 3: View Employee Details

- **UC-ID:** UC-03
- **Title:** View Employee Details
- **Description:** Allows an HR Administrator to view the full details of a specific employee.
- **Actors:** HR Administrator
- **Preconditions:** User is logged in. The employee record exists in the system.
- **Main Flow:**
    1. User navigates to the employee list or search function.
    2. User searches for or selects the desired employee (e.g., by ID or name).

3. User clicks on the employee's record or a 'View' button.
4. System retrieves the employee's full details (ID, name, department, salary, contact details) from the database.
5. System displays the employee's details on the screen in a read-only format.

- **Alternate Flows:**
    1. **A1: Employee Not Found:** If the specified employee cannot be found, the system displays a "Employee not found" message.
- **Postconditions:** The details of the selected employee are displayed to the user. OR An error message is displayed if the employee is not found.

3.2.4 Use Case 4: Update Employee Details

- **UC-ID:** UC-04
- **Title:** Update Employee Details
- **Description:** Allows an HR Administrator to modify the information of an existing employee.
- **Actors:** HR Administrator
- **Preconditions:** User is logged in with privileges to update employees. The employee record exists.
- **Main Flow:**
    1. User finds the employee to update (similar to UC-03, steps 1-2).
    2. User selects an 'Update' or 'Edit' option for that employee.
    3. System displays a form pre-populated with the employee's current details.
    4. User modifies the necessary fields (e.g., department, salary, contact details). The Employee ID might be non-editable.
    5. User submits the updated information.
    6. System validates the modified data.
    7. System saves the updated employee record to the database, overwriting the previous data for modified fields.
    8. System displays a success confirmation message.
- **Alternate Flows:**
    1. **A1: Validation Error:** If validation fails on the modified data, the system displays specific error messages and does not save the changes. The form remains populated.
    2. **A2: Employee Not Found:** If the employee record somehow ceases to exist before saving, display an error.
- **Postconditions:** The employee record is successfully updated in the database. OR The employee record remains unchanged, and error messages are displayed.

3.2.5 Use Case 5: Delete Employee

- **UC-ID:** UC-05
- **Title:** Delete Employee
- **Description:** Allows an HR Administrator to remove an employee record from the system.
- **Actors:** HR Administrator
- **Preconditions:** User is logged in with privileges to delete employees. The employee record exists.
- **Main Flow:**
    1. User finds the employee to delete (similar to UC-03, steps 1-2).
    2. User selects a 'Delete' option for that employee.

3. System prompts the user for confirmation (e.g., "Are you sure you want to delete employee [Name]?").
4. User confirms the deletion.
5. System removes the employee record from the database. (Alternatively, flags the record as inactive).
6. System displays a success confirmation message.

- **Alternate Flows:**
    1. **A1: Deletion Cancelled:** If the user cancels the confirmation prompt, the record is not deleted, and the user is returned to the previous view.
    2. **A2: Employee Not Found:** If the employee record somehow ceases to exist before deletion, display an error.
- **Postconditions:** The employee record is removed from the system (or marked inactive). OR The employee record remains unchanged.

3.2.6 Use Case 6: List All Employees

- **UC-ID:** UC-06
- **Title:** List All Employees
- **Description:** Allows an HR Administrator to view a list of all active employees in the system.
- **Actors:** HR Administrator
- **Preconditions:** User is logged in.
- **Main Flow:**
    1. User navigates to the 'List All Employees' function/page.
    2. System retrieves records for all active employees from the database.
    3. System displays the employees in a list or tabular format, showing key information (e.g., ID, Name, Department).
    4. (Optional) System provides options for sorting or basic filtering of the list.
- **Alternate Flows:**
    1. **A1: No Employees:** If there are no employees in the system, display a message indicating this (e.g., "No employee records found").
- **Postconditions:** A list of all employees is displayed. OR A message indicating no employees exist is shown.

3.2.7 Use Case 7: Generate Department-Wise Report

- **UC-ID:** UC-07
- **Title:** Generate Department-Wise Report
- **Description:** Allows an HR Administrator to generate a report summarizing employee data grouped by department.
- **Actors:** HR Administrator
- **Preconditions:** User is logged in with privileges to run reports. Employee and department data exist.
- **Main Flow:**
    1. User navigates to the 'Reports' section and selects the 'Department-Wise Report' option.
    2. System may prompt the user to select specific departments or generate for all departments.
    3. User confirms the report generation criteria.
    4. System retrieves relevant employee data, grouped and summarized by department.
    5. System generates the report according to the layout specified in Appendix

5.2.1.
    6. System displays the report on screen or provides an option to download it (e.g., as PDF or CSV).
- **Alternate Flows:**
    1. **A1: No Data for Report:** If no employee data exists for the selected criteria, display a message indicating this.
- **Postconditions:** The Department-Wise Report is generated and displayed/made available for download. OR A message indicating no data is available is shown.

4. Non-Functional Requirements

4.1 Performance Requirements

- **NFR-01:** The employee list page shall load within 3 seconds with up to 1000 employee records.
- **NFR-02:** Employee search results shall be returned within 2 seconds.
- **NFR-03:** Saving a new or updated employee record shall complete within 2 seconds under normal load.
- **NFR-04:** Generation of the Department-Wise Report for all employees (up to 1000) shall complete within 10 seconds.
- **NFR-05:** The system shall support up to 20 concurrent HR administrators performing typical tasks without significant performance degradation.

4.2 Security Requirements

- **NFR-06:** All access to the system requires successful user authentication (Ref FR-01).
- **NFR-07:** User passwords must be stored securely using industry-standard hashing algorithms (e.g., bcrypt).
- **NFR-08:** Role-based access control shall be implemented to ensure users can only perform actions and view data appropriate to their role (e.g., HR Admin vs. potential Manager role).
- **NFR-09:** Salary data must be treated as confidential and access restricted appropriately.
- **NFR-10:** The system must be protected against common web application vulnerabilities, including SQL Injection and Cross-Site Scripting (XSS). (Consider OWASP Top 10).
- **NFR-11:** (Optional) Session management shall be secure, including session timeouts after a period of inactivity (e.g., 30 minutes).

4.3 Usability Requirements

- **NFR-12:** The user interface shall be intuitive and consistent across all modules/pages (e.g., consistent navigation, button placement, terminology).
- **NFR-13:** The system shall provide clear feedback messages for user actions (e.g., success confirmations, error messages, warnings).
- **NFR-14:** Data entry forms shall be easy to understand and use. Required fields must be clearly marked. Error messages should guide the user to correct input.
- **NFR-15:** The time to train a new HR Administrator with basic computer skills to use the core functions (CRUD, List, Report) should not exceed 2 hours.
- **NFR-16:** (Optional) The system should adhere to basic web accessibility guidelines (e.g., WCAG 2.1 Level A).

4.4 Reliability Requirements

- **NFR-17:** The system shall have a target availability of 99.5% during standard

BitFutura business hours (e.g., Monday-Friday, 9 AM - 6 PM local time).
- **NFR-18:** Automated database backups shall be performed daily, with a defined retention policy.
- **NFR-19:** The system must handle errors gracefully, providing informative error messages to the user rather than crashing or showing technical stack traces.
- **NFR-20:** Data integrity must be maintained; actions like deleting a department should handle associated employees appropriately (e.g., prevent deletion if employees are assigned, or require reassignment). (This might also be a functional requirement depending on how it's handled).

5. Appendices

5.1 Glossary

Refer to Section 1.3 Definitions, Acronyms, and Abbreviations.

5.2 Additional Information

5.2.1 Department-Wise Report Layout Design

TBC

Report Title: Employee Summary by Department

Generated On: [Date and Time]

Structure: The report will be grouped by Department Name. For each department, a table will list the employees belonging to it.

Department Header:

- Department Name: [Name of the Department]

Employee Table (within each Department section):

- **Columns:**
  - Employee ID
  - Employee Name
  - Salary (Note: Consider if salary should be in this report or if it's too sensitive - clarify if possible)
  - Contact Details (e.g., Email or Phone)
- **Sorting:** Employees within each department section should be listed alphabetically by Employee Name.

Department Footer

- Total Employee Count for Department: [Number]
- Total Salary for Department: [Sum of Salaries in this Dept] (If Salary included)

Overall Report Footer

- Total Number of Employees across all listed departments.
- Total Salary across all listed departments.

Format: The report should be viewable online and downloadable, preferably in CSV format for data manipulation and potentially PDF format for printing/archiving.

5.2.2 Flowchart

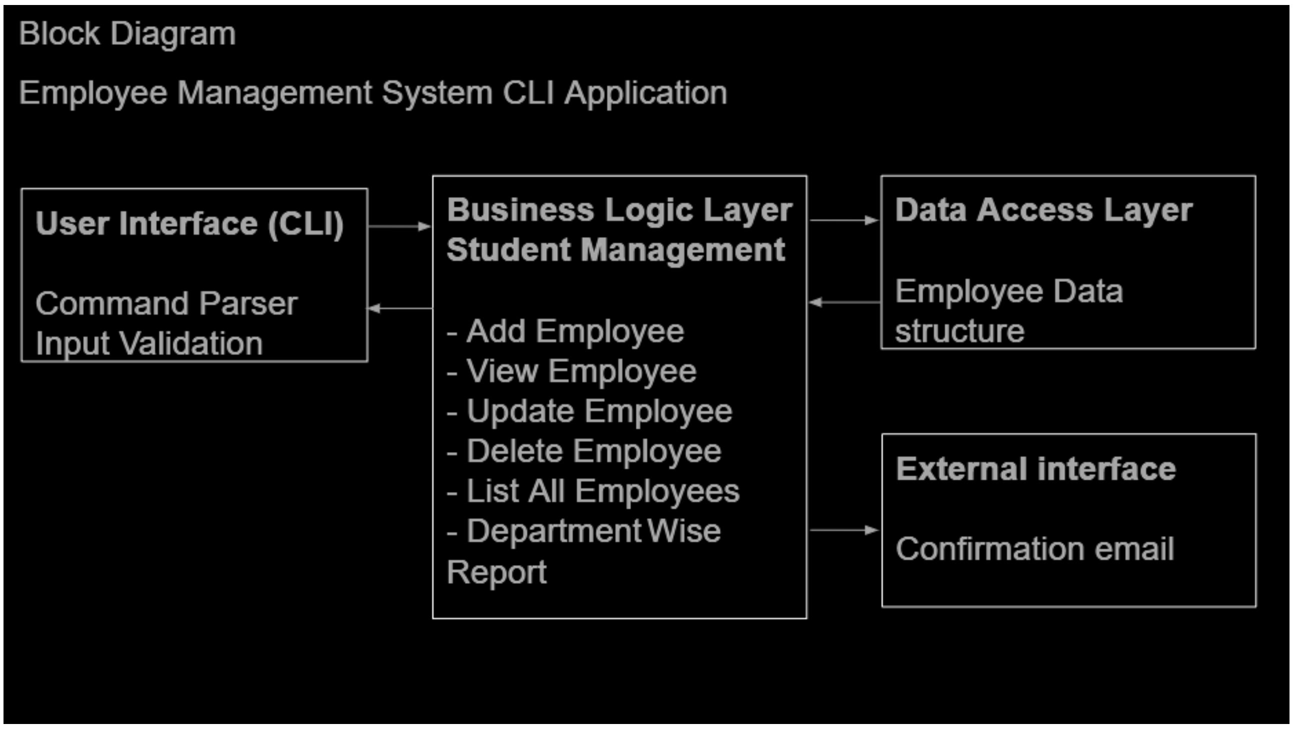
Please copy and paste the code from the file flowchart.mmd into https://mermaid.live/ to view flowchart.

6. Bibliography

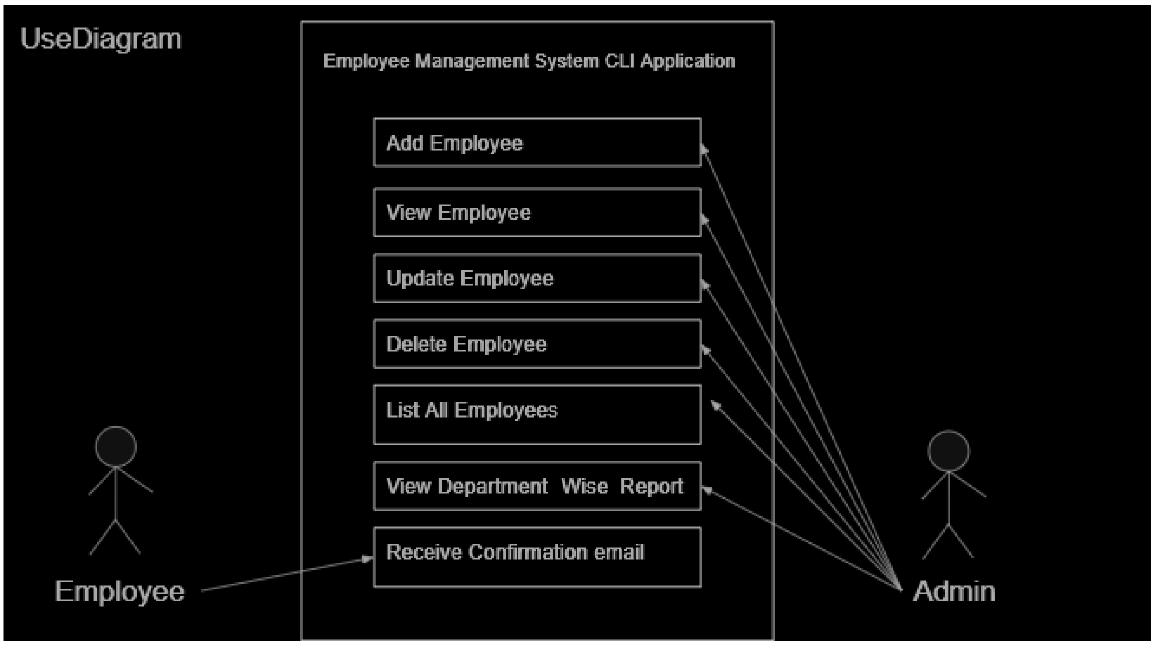
Lithan Academy (2025) Employee Management System Assignment Brief. PDWD-PFS-FA01. Singapore: Lithan Academy.
Lithan Academy (2025) How to write system specifications. PDWD-PFS-FC01-Additional_Content-01. Singapore: Lithan Academy.

2. Design Diagrams

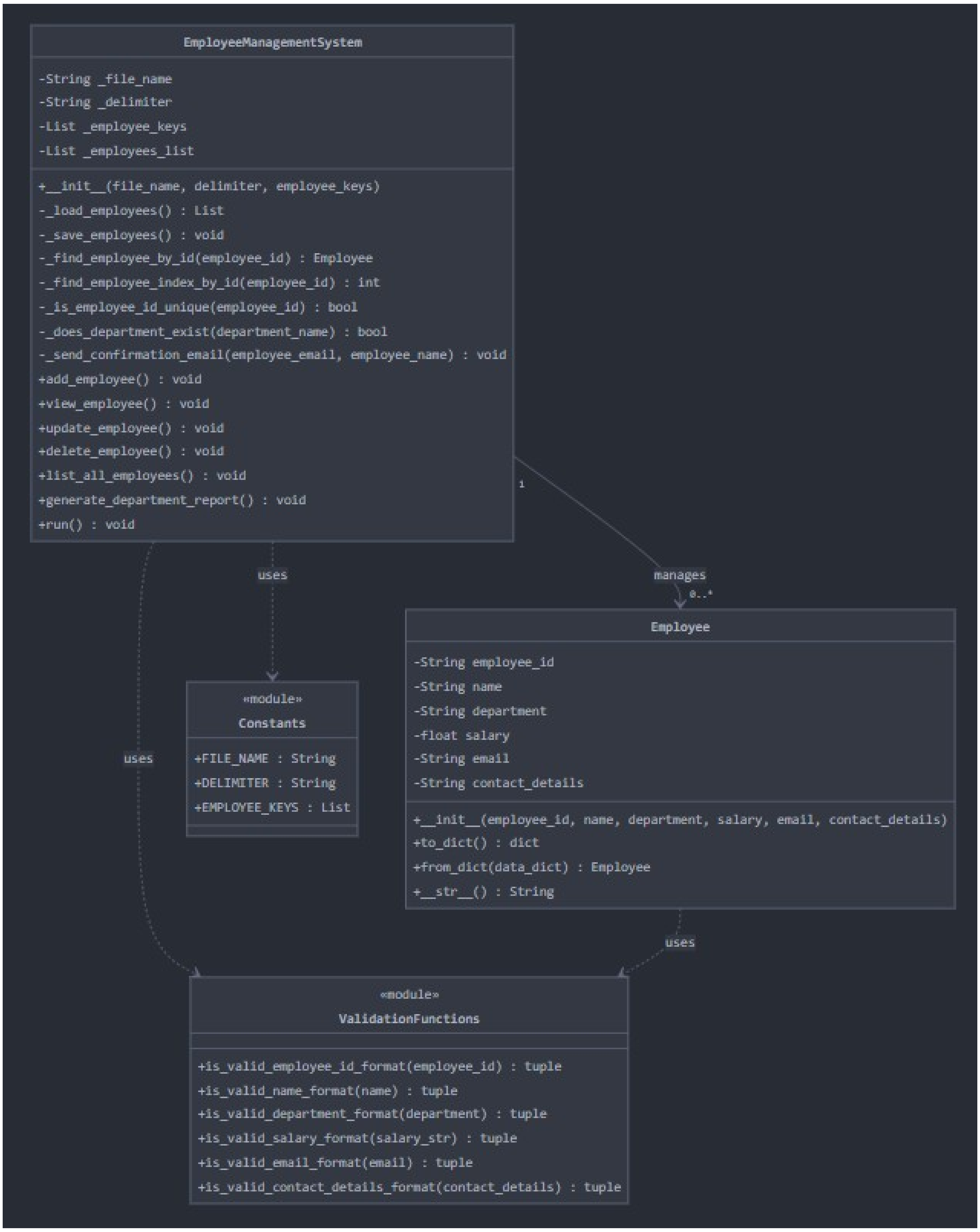
System Block Diagram



Use-case Diagram

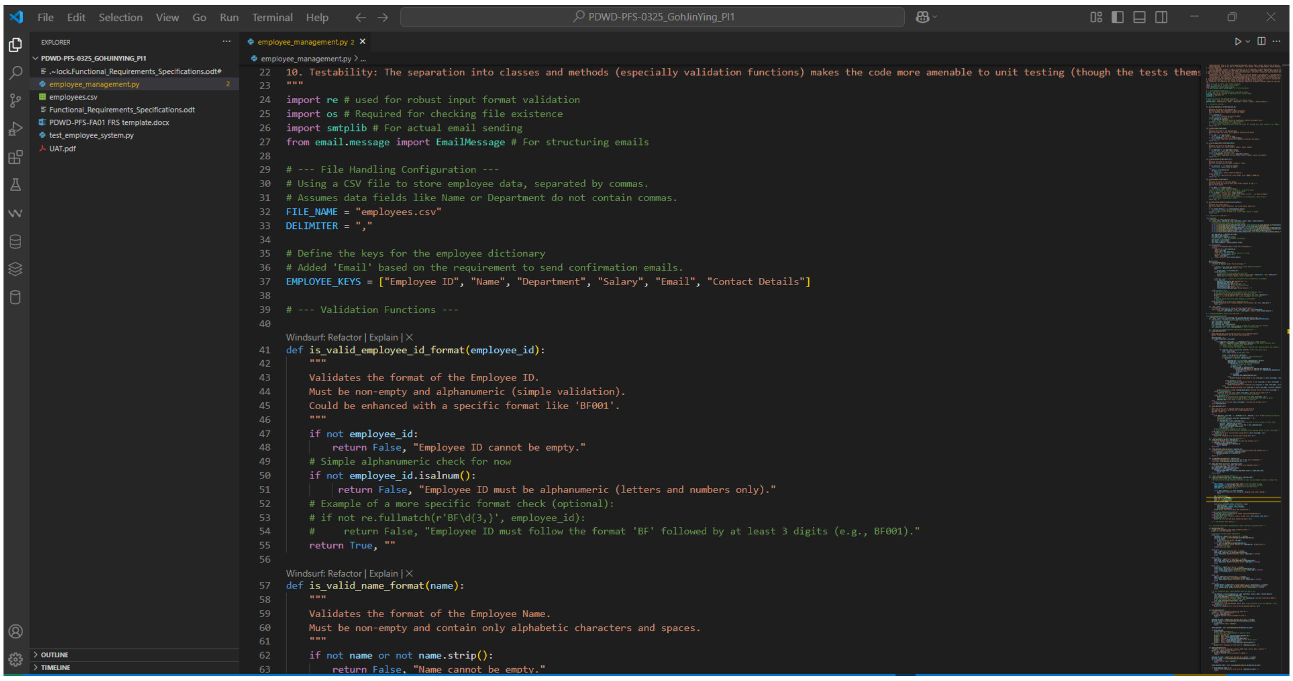


Class Diagram


EmployeeManagementSystem
-String _file_name
-String _delimiter
-List _employee_keys
-List _employees_list
+__init__(file_name, delimiter, employee_keys)
-_load_employees() : List
-_save_employees() : void
-_find_employee_by_id(employee_id) : Employee
-_find_employee_index_by_id(employee_id) : int
-_is_employee_id_unique(employee_id) : bool
-_does_department_exist(department_name) : bool
-_send_confirmation_email(employee_email, employee_name) : void
+add_employee() : void
+view_employee() : void
+update_employee() : void
+delete_employee() : void
+list_all_employees() : void
+generate_department_report() : void
+run() : void
1
manages
0..*
uses
uses
«module»
Constants
+FILE_NAME : String
+DELIMITER : String
+EMPLOYEE_KEYS : List
Employee
-String employee_id
-String name
-String department
-float salary
-String email
-String contact_details
+__init__(employee_id, name, department, salary, email, contact_details)
+to_dict() : dict
+from_dict(data_dict) : Employee
+__str__() : String
uses
«module»
ValidationFunctions
+is_valid_employee_id_format(employee_id) : tuple
+is_valid_name_format(name) : tuple
+is_valid_department_format(department) : tuple
+is_valid_salary_format(salary_str) : tuple
+is_valid_email_format(email) : tuple
+is_valid_contact_details_format(contact_details) : tuple

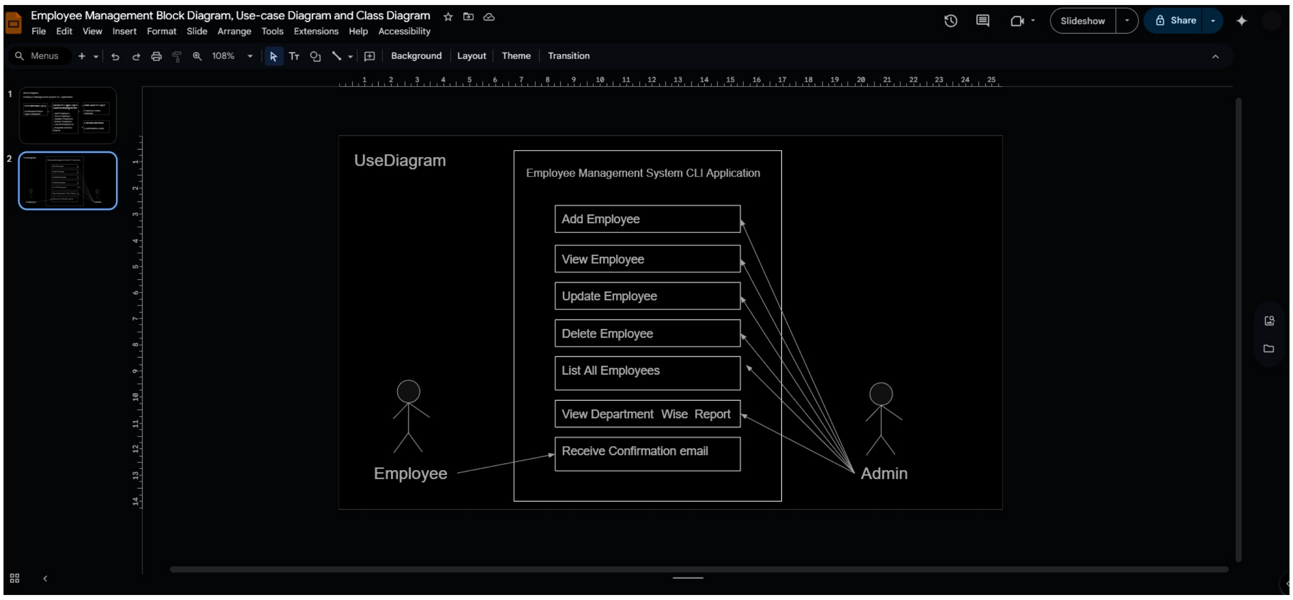
3. Tools Used

For Application


File Edit Selection View Go Run Terminal Help
PDWD-PFS-0325_GohJinYing_PI1
EXPLORER
PDWD-PFS-0325_GOHJINYING_PI1
.~lock.Functional_Requirements_Specifications.odt#
employee_management.py
employees.csv
Functional_Requirements_Specifications.odt
PDWD-PFS-FA01 FRS template.docx
test_employee_system.py
UAT.pdf
OUTLINE
TIMELINE

```python
10. Testability: The separation into classes and methods (especially validation functions) makes the code more amenable to unit testing (though the tests them
"""
import re # used for robust input format validation
import os # Required for checking file existence
import smtplib # For actual email sending
from email.message import EmailMessage # For structuring emails

# --- File Handling Configuration ---
# Using a CSV file to store employee data, separated by commas.
# Assumes data fields like Name or Department do not contain commas.
FILE_NAME = "employees.csv"
DELIMITER = ","

# Define the keys for the employee dictionary
# Added 'Email' based on the requirement to send confirmation emails.
EMPLOYEE_KEYS = ["Employee ID", "Name", "Department", "Salary", "Email", "Contact Details"]

# --- Validation Functions ---

def is_valid_employee_id_format(employee_id):
    """
    Validates the format of the Employee ID.
    Must be non-empty and alphanumeric (simple validation).
    Could be enhanced with a specific format like 'BF001'.
    """
    if not employee_id:
        return False, "Employee ID cannot be empty."
    # Simple alphanumeric check for now
    if not employee_id.isalnum():
        return False, "Employee ID must be alphanumeric (letters and numbers only)."
    # Example of a more specific format check (optional):
    # if not re.fullmatch(r'BF\d{3,}', employee_id):
    #     return False, "Employee ID must follow the format 'BF' followed by at least 3 digits (e.g., BF001)."
    return True, ""

def is_valid_name_format(name):
    """
    Validates the format of the Employee Name.
    Must be non-empty and contain only alphabetic characters and spaces.
    """
    if not name or not name.strip():
        return False, "Name cannot be empty."
```

For Diagrams


Employee Management Block Diagram, Use-case Diagram and Class Diagram
File Edit View Insert Format Slide Arrange Tools Extensions Help Accessibility
Slideshow
Share
Background Layout Theme Transition
UseDiagram
Employee Management System CLI Application
Add Employee
View Employee
Update Employee
Delete Employee
List All Employees
View Department Wise Report
Receive Confirmation email
Employee
Admin

4. Controls, Elements and Features

Data structures used for fields (in table format)

The fields are primarily defined by the EMPLOYEE_KEYS and how they are handled within the Employee class and the CSV file.

Field Name	In-Memory Datatype (Python Employee object)	CSV File Datatype (How it's stored)	Notes
Employee ID	str	string	Alphanumeric, validated by is_valid_employee_id_format.
Name	str	string	Alphabetic characters and spaces, validated by is_valid_name_format.
Department	str	string	Alphanumeric, spaces, hyphens, validated by is_valid_department_format.
Salary	float	string (representing a number)	Stored as a float in memory for calculations, written as a formatted string (e.g., "50000.00") to CSV. Validated by is_valid_salary_format.
Email	str	string	Validated for basic email structure by is_valid_email_format.
Contact Details	str	string	General string, validated for non-emptiness by is_valid_contact_details_format.

How classes created meet the project requirements

The primary classes developed for this system are:

- **Employee Class:**
    - **Purpose:** To model a single employee record. It encapsulates all the attributes of an employee (ID, name, department, salary, email, contact details).
    - **Functional Requirements Met:**
        - Provides a structure for storing individual employee data, which is fundamental for all CRUD operations (Add, View, Update, Delete Employee).
        - Includes validation within its __init__ method, ensuring data integrity when an employee object is created or loaded.
        - to_dict(): Converts employee object data into a dictionary, facilitating easy serialization for saving to the CSV file (Interface: File Handling).

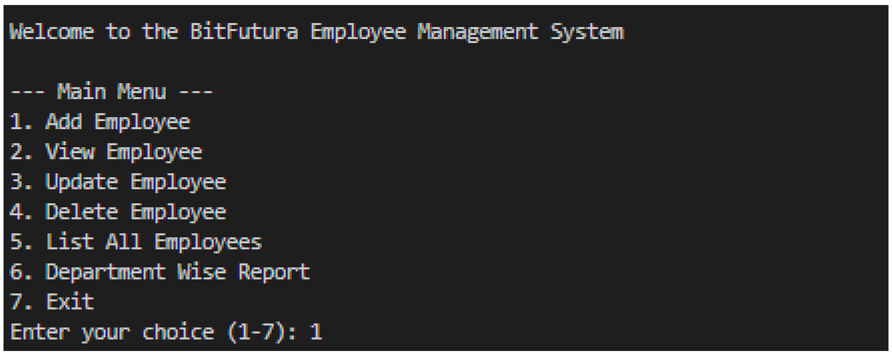
    - from_dict(): A static method to create an Employee object from a dictionary (typically read from the CSV file), facilitating deserialization (Interface: File Handling).
    - __str__(): Provides a user-friendly string representation, useful for debugging or simple display.
- **EmployeeManagementSystem Class:**
  - **Purpose:** To manage the entire collection of Employee objects and orchestrate the application's logic. It acts as the main controller for the system.
  - **Functional Requirements Met:**
    - **CRUD Operations:**
      - add_employee(): Implements "Add Employee" requirement, including input prompting and validation.
      - view_employee(): Implements "View Employee" requirement.
      - update_employee(): Implements "Update Employee" requirement, including input for new values and validation.
      - delete_employee(): Implements "Delete Employee" requirement with confirmation.
    - **Listing & Reporting:**
      - list_all_employees(): Implements "List All Employees" requirement.
      - generate_department_report(): Implements "Department Wise Report" requirement, including the design of the report layout.
    - **Email Confirmation:**
      - _send_confirmation_email(): (Called by add_employee()) Implements "Send confirmation email" requirement (currently a simulation but structured for real implementation).
    - **File Handling (Interface):**
      - _load_employees(): Loads employee data from employees.csv on startup.
      - _save_employees(): Saves employee data to employees.csv after modifications or on exit.
    - **Command-Line Interface (Interface):**
      - run(): Provides the main menu-driven command-line interface for user interaction.
    - **OOP Principles:**
      - Demonstrates encapsulation (hiding _employees_list, _file_name, internal methods) and abstraction (providing high-level methods for users).
    - **Helper/Internal Methods:**
      - _find_employee_by_id(), _find_employee_index_by_id(), _is_employee_id_unique(), _does_department_exist():

Internal utility methods to support the main functionalities, promoting code reusability and clarity.

- **EmailMessage Class (from email.message module - an external library class, but used to meet a requirement):**
    - **Purpose:** Used within the _send_confirmation_email method of the EmployeeManagementSystem class to construct the email message (subject, sender, recipient, body).
    - **Functional Requirements Met:**
        - Essential for structuring the email content as part of the "Send confirmation email" requirement.

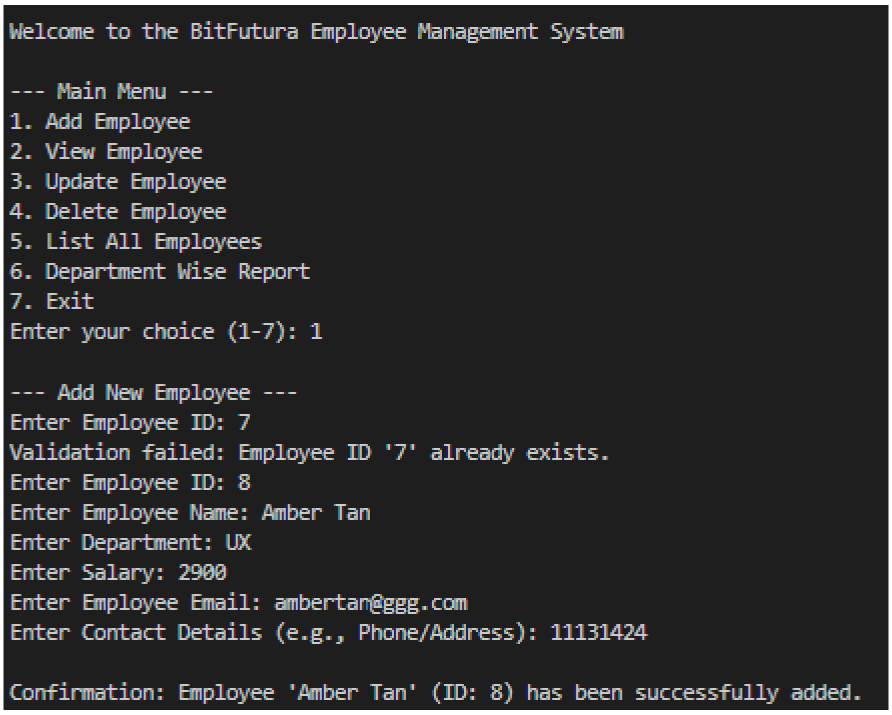
5. Development

Main menu

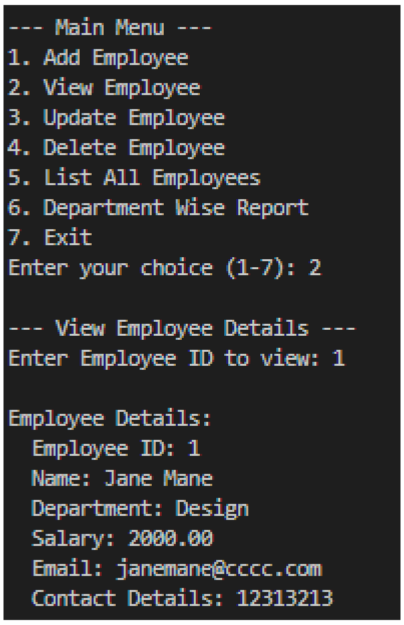


```
Welcome to the BitFutura Employee Management System

--- Main Menu ---
1. Add Employee
2. View Employee
3. Update Employee
4. Delete Employee
5. List All Employees
6. Department Wise Report
7. Exit
Enter your choice (1-7): 1
```

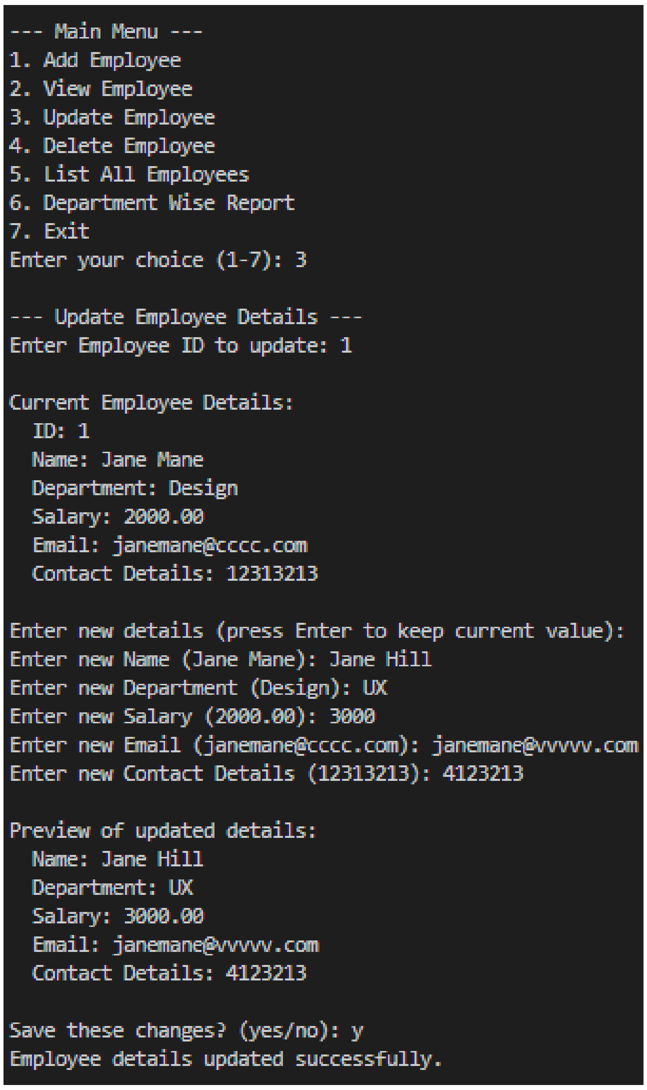
CRUD



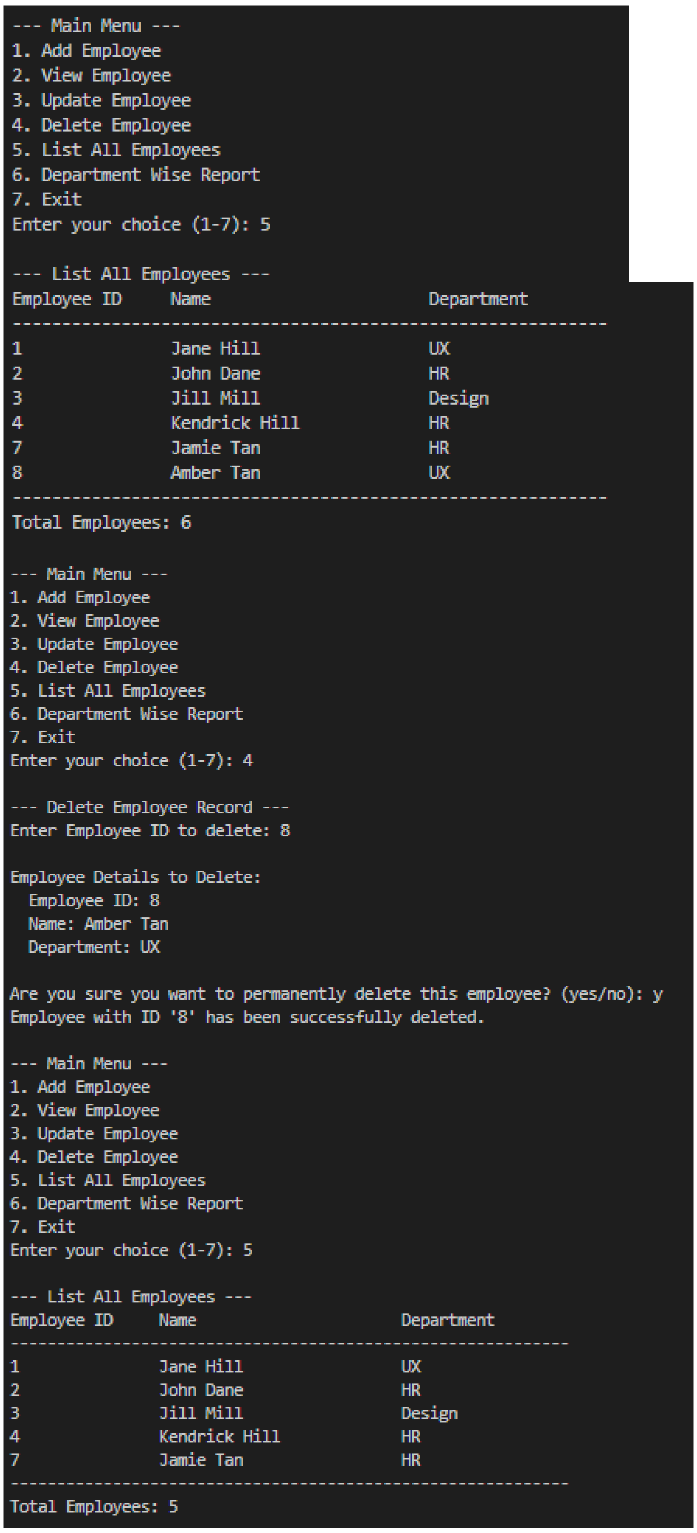
```
Welcome to the BitFutura Employee Management System

--- Main Menu ---
1. Add Employee
2. View Employee
3. Update Employee
4. Delete Employee
5. List All Employees
6. Department Wise Report
7. Exit
Enter your choice (1-7): 1

--- Add New Employee ---
Enter Employee ID: 7
Validation failed: Employee ID '7' already exists.
Enter Employee ID: 8
Enter Employee Name: Amber Tan
Enter Department: UX
Enter Salary: 2900
Enter Employee Email: ambertan@ggg.com
Enter Contact Details (e.g., Phone/Address): 11131424

Confirmation: Employee 'Amber Tan' (ID: 8) has been successfully added.
```



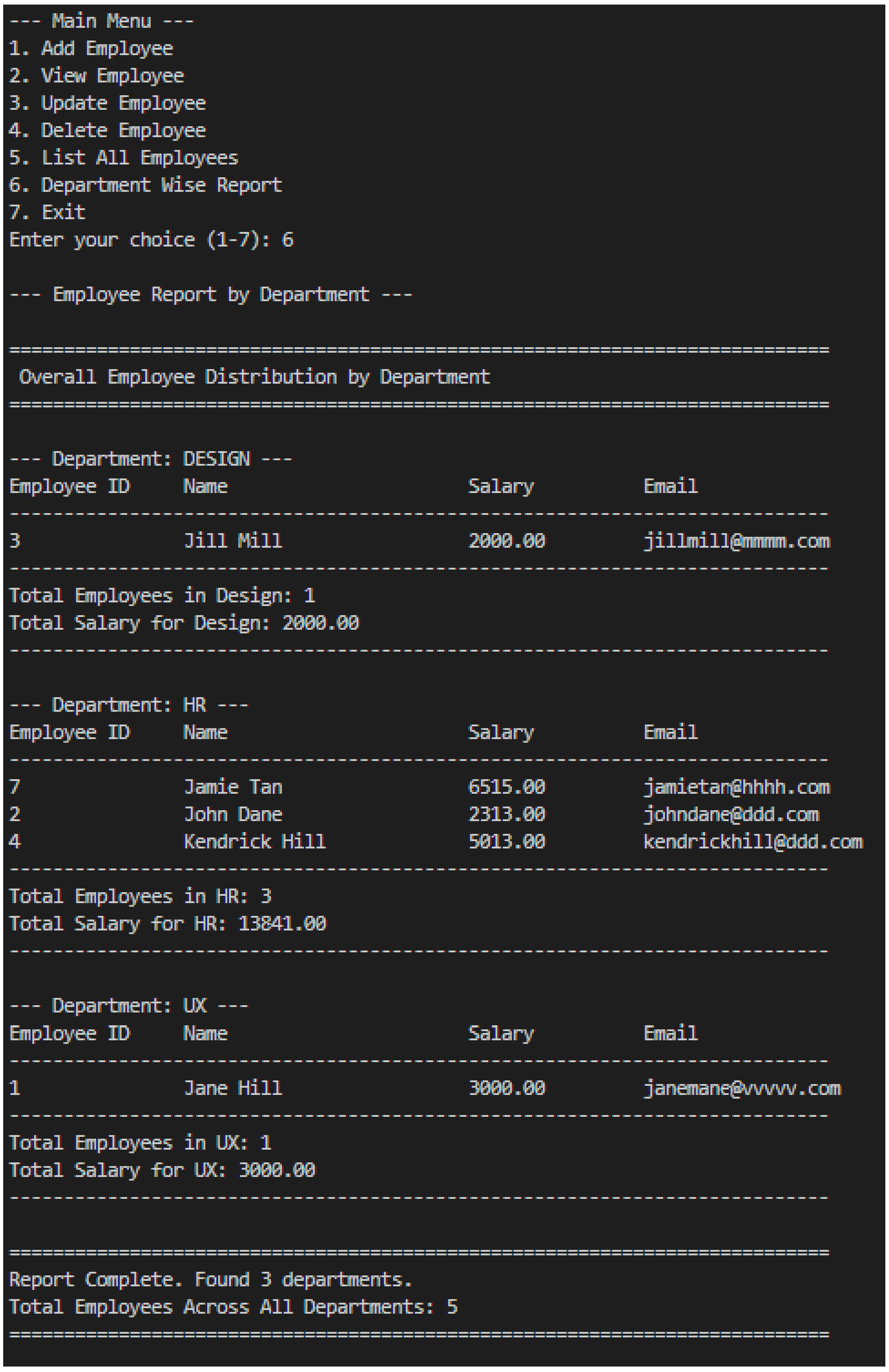
```
--- Main Menu ---
1. Add Employee
2. View Employee
3. Update Employee
4. Delete Employee
5. List All Employees
6. Department Wise Report
7. Exit
Enter your choice (1-7): 2

--- View Employee Details ---
Enter Employee ID to view: 1

Employee Details:
  Employee ID: 1
  Name: Jane Mane
  Department: Design
  Salary: 2000.00
  Email: janemane@cccc.com
  Contact Details: 12313213
```



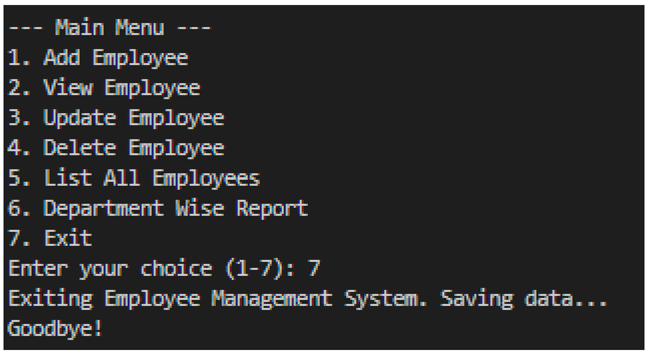
```
--- Main Menu ---
1. Add Employee
2. View Employee
3. Update Employee
4. Delete Employee
5. List All Employees
6. Department Wise Report
7. Exit
Enter your choice (1-7): 3

--- Update Employee Details ---
Enter Employee ID to update: 1

Current Employee Details:
  ID: 1
  Name: Jane Mane
  Department: Design
  Salary: 2000.00
  Email: janemane@cccc.com
  Contact Details: 12313213

Enter new details (press Enter to keep current value):
Enter new Name (Jane Mane): Jane Hill
Enter new Department (Design): UX
Enter new Salary (2000.00): 3000
Enter new Email (janemane@cccc.com): janemane@vvvvv.com
Enter new Contact Details (12313213): 4123213

Preview of updated details:
  Name: Jane Hill
  Department: UX
  Salary: 3000.00
  Email: janemane@vvvvv.com
  Contact Details: 4123213

Save these changes? (yes/no): y
Employee details updated successfully.
```



```
--- Main Menu ---
1. Add Employee
2. View Employee
3. Update Employee
4. Delete Employee
5. List All Employees
6. Department Wise Report
7. Exit
Enter your choice (1-7): 5

--- List All Employees ---
Employee ID     Name                      Department
------------------------------------------------------------
1               Jane Hill                 UX
2               John Dane                 HR
3               Jill Mill                 Design
4               Kendrick Hill             HR
7               Jamie Tan                 HR
8               Amber Tan                 UX
------------------------------------------------------------
Total Employees: 6

--- Main Menu ---
1. Add Employee
2. View Employee
3. Update Employee
4. Delete Employee
5. List All Employees
6. Department Wise Report
7. Exit
Enter your choice (1-7): 4

--- Delete Employee Record ---
Enter Employee ID to delete: 8

Employee Details to Delete:
  Employee ID: 8
  Name: Amber Tan
  Department: UX

Are you sure you want to permanently delete this employee? (yes/no): y
Employee with ID '8' has been successfully deleted.

--- Main Menu ---
1. Add Employee
2. View Employee
3. Update Employee
4. Delete Employee
5. List All Employees
6. Department Wise Report
7. Exit
Enter your choice (1-7): 5

--- List All Employees ---
Employee ID     Name                      Department
------------------------------------------------------------
1               Jane Hill                 UX
2               John Dane                 HR
3               Jill Mill                 Design
4               Kendrick Hill             HR
7               Jamie Tan                 HR
------------------------------------------------------------
Total Employees: 5
```

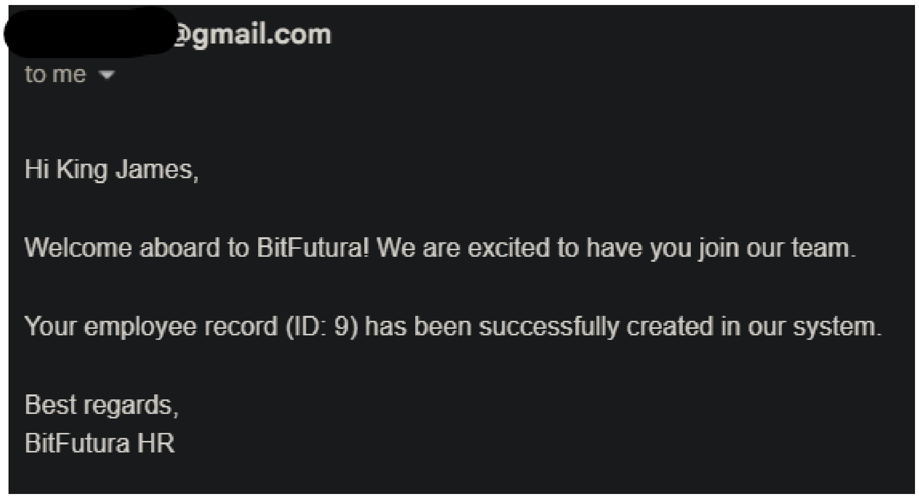


```
--- Main Menu ---
1. Add Employee
2. View Employee
3. Update Employee
4. Delete Employee
5. List All Employees
6. Department Wise Report
7. Exit
Enter your choice (1-7): 6

--- Employee Report by Department ---

===========================================================================
 Overall Employee Distribution by Department
===========================================================================

--- Department: DESIGN ---
Employee ID     Name                      Salary          Email
---------------------------------------------------------------------------
3               Jill Mill                 2000.00         jillmill@mmmm.com
---------------------------------------------------------------------------
Total Employees in Design: 1
Total Salary for Design: 2000.00
---------------------------------------------------------------------------

--- Department: HR ---
Employee ID     Name                      Salary          Email
---------------------------------------------------------------------------
7               Jamie Tan                 6515.00         jamietan@hhhh.com
2               John Dane                 2313.00         johndane@ddd.com
4               Kendrick Hill             5013.00         kendrickhill@ddd.com
---------------------------------------------------------------------------
Total Employees in HR: 3
Total Salary for HR: 13841.00
---------------------------------------------------------------------------

--- Department: UX ---
Employee ID     Name                      Salary          Email
---------------------------------------------------------------------------
1               Jane Hill                 3000.00         janemane@vvvvv.com
---------------------------------------------------------------------------
Total Employees in UX: 1
Total Salary for UX: 3000.00
---------------------------------------------------------------------------


===========================================================================
Report Complete. Found 3 departments.
Total Employees Across All Departments: 5
===========================================================================
```



```
--- Main Menu ---
1. Add Employee
2. View Employee
3. Update Employee
4. Delete Employee
5. List All Employees
6. Department Wise Report
7. Exit
Enter your choice (1-7): 7
Exiting Employee Management System. Saving data...
Goodbye!
```

Email



[redacted]@gmail.com
to me

Hi King James,

Welcome aboard to BitFutura! We are excited to have you join our team.

Your employee record (ID: 9) has been successfully created in our system.

Best regards,
BitFutura HR

6. Testing

Unit Test – Test Cases

Please refer to file test_employee_system.py

Unit Test – Test Results

```
F.........No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
FNo data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
FNo data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
```

```
FNo data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.......
======================================================================
FAIL: test_employee_from_dict_invalid_salary
(__main__.TestEmployeeClass.test_employee_from_dict_invalid_salary)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:\Users\justj\Downloads\Lithan Module 2\0 Project Presentation\Source Code\
test_employee_system.py", line 178, in test_employee_from_dict_invalid_salary
    mock_print.assert_called_once_with(f"Error creating employee from dictionary data: could not
convert string to float: 'not-a-number'. Data: {invalid_data}")

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\justj\AppData\Local\Programs\Python\Python313\Lib\unittest\mock.py", line 991, in
assert_called_once_with
    return self.assert_called_with(*args, **kwargs)
           ~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^
  File "C:\Users\justj\AppData\Local\Programs\Python\Python313\Lib\unittest\mock.py", line 979, in
assert_called_with
    raise AssertionError(_error_message()) from cause
AssertionError: expected call not found.
Expected: print("Error creating employee from dictionary data: could not convert string to float:
'not-a-number'. Data: {'Employee ID': 'BF001', 'Name': 'Alice Smith', 'Department': 'Engineering',
'Salary': 'not-a-number', 'Email': 'alice.s@bitfutura.test', 'Contact Details': '123-456-7890'}")
  Actual: print("Error creating employee: Invalid salary format 'not-a-number'. Data: {'Employee ID':
'BF001', 'Name': 'Alice Smith', 'Department': 'Engineering', 'Salary': 'not-a-number', 'Email':
'alice.s@bitfutura.test', 'Contact Details': '123-456-7890'}")

======================================================================
FAIL: test_save_employees (__main__.TestEmployeeManagementSystem.test_save_employees)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\justj\AppData\Local\Programs\Python\Python313\Lib\unittest\mock.py", line 1426,
in patched
    return func(*newargs, **newkeywargs)
  File "c:\Users\justj\Downloads\Lithan Module 2\0 Project Presentation\Source Code\
test_employee_system.py", line 338, in test_save_employees
    handle.write.assert_any_call(expected_line2)
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^
  File "C:\Users\justj\AppData\Local\Programs\Python\Python313\Lib\unittest\mock.py", line 1050,
in assert_any_call
```

```
    raise AssertionError(
        '%s call not found' % expected_string
    ) from cause
AssertionError: write('BF002,Bob Johnson,Marketing,60000,bob.j@bitfutura.test,987-654-3210\n')
call not found

======================================================================
FAIL: test_send_email_auth_error
(__main__.TestEmployeeManagementSystem.test_send_email_auth_error)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\justj\AppData\Local\Programs\Python\Python313\Lib\unittest\mock.py", line 1426,
in patched
    return func(*newargs, **newkeywargs)
  File "c:\Users\justj\Downloads\Lithan Module 2\0 Project Presentation\Source Code\
test_employee_system.py", line 449, in test_send_email_auth_error
    mock_print.assert_any_call("Email Error: Authentication failed. Check username/password.")

~~~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^
  File "C:\Users\justj\AppData\Local\Programs\Python\Python313\Lib\unittest\mock.py", line 1050,
in assert_any_call
    raise AssertionError(
        '%s call not found' % expected_string
    ) from cause
AssertionError: print('Email Error: Authentication failed. Check username/password.') call not found

======================================================================
FAIL: test_send_email_success
(__main__.TestEmployeeManagementSystem.test_send_email_success)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\justj\AppData\Local\Programs\Python\Python313\Lib\unittest\mock.py", line 1426,
in patched
    return func(*newargs, **newkeywargs)
  File "c:\Users\justj\Downloads\Lithan Module 2\0 Project Presentation\Source Code\
test_employee_system.py", line 418, in test_send_email_success
    mock_print.assert_any_call(f"Simulating email send to {test_email} with subject 'Employee Action
Confirmation'")

~~~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\justj\AppData\Local\Programs\Python\Python313\Lib\unittest\mock.py", line 1050,
in assert_any_call
    raise AssertionError(
        '%s call not found' % expected_string
    ) from cause
```

```
AssertionError: print("Simulating email send to recipient@example.com with subject 'Employee
Action Confirmation'") call not found

----------------------------------------------------------------------
Ran 47 tests in 0.036s

FAILED (failures=4)
```

User Acceptance Test – Test Cases & Results

User Acceptance Test (UAT) Plan for Employee Management System

1 Overview

This UAT plan verifies the functionality of the Employee Management System (EMS) for BitFutura, ensuring it meets the specified requirements for managing employee records via a command-line interface. The system supports CRUD operations, employee listing, department wise reporting, email confirmation, and persistent storage using a CSV file. The tests focus on usability, data integrity, and system reliability for end-users (HR staff).

2 TestScope

- System: Command-line Employee Management System
- Users: HR personnel managing employee records
- Features Tested:
  - AddEmployee (with validation and email confirmation)
  - View Employee
  - Update Employee
  - Delete Employee
  - List All Employees
  - Department-Wise Report
  - File Handling (CSV persistence)
  - MenuNavigation and Error Handling
- Exclusions: Unit tests, performance testing, actual email sending (simulated)

3 TestEnvironment

- Platform: Python 3.8+ on a local machine
- Dependencies: Standard libraries (re, os, smtplib, email.message)
- Data: Test data in employees.csv (created during testing)
- Prerequisites:
  - Python installed
  - Write permissions for CSV file in the working directory 1
  - Clean employees.csv file (or none) before starting

4 TestScenarios

4.1 AddEmployee

Objective: Verify that users can add a new employee with valid data, data is saved to CSV, and a confirmation email is simulated.

4.1.1 Test Case ID: UAT-ADD-01

Steps:

1. Run the EMS and select option 1 (Add Employee).
2. Enter:
   - Employee ID: “EMP001”
   - Name: “John Doe”

- Department: “Engineering”
- Salary: “75000”
- Email: “john.doe@example.com”
- Contact Details: “123-456-7890”

3. Confirm the addition.
4. Check employees.csv for the new record.

Expected Outcome:

- Success message: “Employee ‘John Doe’ (ID: EMP001) has been successfully added.”
- Simulated email confirmation message displayed.
- employees.csv contains the new employee record with all fields.

4.1.2 Test Case ID: UAT-ADD-02 (Invalid Input)

Steps:

1. Select option
1. 2. Enter invalid data:
   - Employee ID: “” (empty)
   - Name: “John123” (invalid characters) 2
   - Salary: “-5000” (negative)
   - Email: “invalid email” (invalid format)
3. Attempt to proceed.

Expected Outcome:

- Validation error messages for each invalid input (e.g., “Employee ID cannot be empty”).
- Employee is not added; no changes in employees.csv.

4.2 ViewEmployee

Objective: Ensure users can view an employee’s details by ID.

4.2.1 Test Case ID: UAT-VIEW-01

Steps:

1. Add an employee (e.g., ID: “EMP002”, Name: “Jane Smith”).
2. Select option 2 (View Employee).
3. Enter Employee ID: “EMP002”.

Expected Outcome:

- Displays details: ID, Name, Department, Salary, Email, Contact Details.
- Format matches: “Employee ID: EMP002”, “Name: Jane Smith”, etc.

4.2.2 Test Case ID: UAT-VIEW-02 (Non-existent ID)

Steps:

1. Select option 2.
2. Enter Employee ID: “EMP999”.

Expected Outcome:

- Error message: “Error: Employee not found with ID ‘EMP999’.”

4.3 Update Employee

Objective: Confirm users can update employee details with validation and save changes.

4.3.1 Test Case ID: UAT-UPDATE-01

Steps:

1. Add an employee (ID: “EMP003”, Name: “Alice Brown”, Department: “HR”, Salary: “60000”).
2. Select option 3 (Update Employee).
3. Enter ID: “EMP003”.
4. Update:
   - Department: “Marketing”

        - Salary: “65000”
        - Leave other fields unchanged (press Enter).
    5. Confirm changes (yes).
    6. View the employee to verify.
    7. Check employees.csv.

Expected Outcome:

- Success message: “Employee details updated successfully.”
- View shows updated Department (“Marketing”) and Salary (“65000”).
- employees.csv reflects changes.

4.3.2 Test Case ID: UAT-UPDATE-02 (Cancel Update)

Steps:

1. Select option 3, enter ID: “EMP003”.
2. Make changes (e.g., Salary: “70000”).
3. Cancel changes (no).

Expected Outcome:

- Message: “Update cancelled. No changes were saved.”
- Nochanges in employees.csv.

4.4 Delete Employee

Objective: Verify users can delete an employee after confirmation.

4.4.1 Test Case ID: UAT-DELETE-01

Steps:

1. Add an employee (ID: “EMP004”).
2. Select option 4 (Delete Employee).
3. Enter ID: “EMP004”.
4. Confirm deletion (yes).
5. Check employees.csv.

Expected Outcome:

- Success message: “Employee with ID ‘EMP004’ has been successfully deleted.”
- Employee is removed from employees.csv.

4.4.2 Test Case ID: UAT-DELETE-02 (Non-existent ID)

Steps:

1. Select option 4, enter ID: “EMP999”.
2. Confirm deletion (yes).

Expected Outcome:

- Error message: “Error: Employee not found with ID ‘EMP999’.”

4.5 List All Employees

Objective: Ensure users can view a formatted list of all employees.

4.5.1 Test Case ID: UAT-LIST-01

Steps:

1. Add multiple employees (e.g., EMP001, EMP002, EMP003).
2. Select option 5 (List All Employees).

Expected Outcome:

- Displays a table with columns: Employee ID, Name, Department.
- Lists all employees in sorted order by ID.
- Shows total employee count.

4.5.2 Test Case ID: UAT-LIST-02 (Empty List)

Steps:

1. Delete all employees or start with an empty employees.csv. 5

2. Select option 5.

Expected Outcome:

• Message: “No employee records found.”

4.6 Department-Wise Report

Objective: Verify the system generates a department-wise employee report.

4.6.1 Test Case ID: UAT-REPORT-01

Steps:

1. Add employees in different departments (e.g., 2 in “Engineering”, 1 in “HR”).
2. Select option 6 (Department Wise Report).

Expected Outcome:

• Displays sections for each department (e.g., “ENGINEERING”, “HR”).
• Each section lists: Employee ID, Name, Salary, Email.
• Shows total employees and total salary per department.
• Summary includes total departments and total employees.

4.6.2 Test Case ID: UAT-REPORT-02 (No Employees)

Steps:

1. Start with an empty employee list.
2. Select option 6.

Expected Outcome:

• Message: “No employee records available to generate reports.”

4.7 File Handling

Objective: Confirm data persistence in employees.csv.

4.7.1 Test Case ID: UAT-FILE-01

Steps:

1. Add two employees.
2. Exit the system (option 7).
3. Restart the EMS and select option 5 (List All Employees).

Expected Outcome:

• Previously added employees are listed.
• employees.csv contains all records with correct headers and data.

4.7.2 Test Case ID: UAT-FILE-02 (File Not Found)

Steps:

1. Delete employees.csv if it exists.
2. Start the EMS.

Expected Outcome:

• Message: “No data file found (employees.csv). Starting with an empty list.”
• System runs without crashing.

4.8 MenuNavigation and Error Handling

Objective: Ensure the CLI is user-friendly and handles errors gracefully.

4.8.1 Test Case ID: UAT-MENU-01

Steps:

1. Start the EMS.
2. Enter invalid menu choice (e.g., “8”).
3. Enter non-numeric input (e.g., “abc”).

Expected Outcome:

• Message: “Invalid choice. Please enter a number between 1 and 7.”
• Returns to main menu without crashing.

4.8.2 Test Case ID: UAT-MENU-02 (Interrupt)

Steps:

1. During an operation (e.g., Add Employee), press Ctrl+C.

Expected Outcome:

- Message: “Operation cancelled by user. Returning to main menu.”
- System remains operational. 7

5 Acceptance Criteria

- All test cases marked UAT-*-01 pass (primary functionality).
- Invalid inputs and edge cases (UAT-*-02) are handled with appropriate error messages.
- Data persists correctly in employees.csv.
- Nosystem crashes during testing.
- CLI is intuitive, with clear prompts and feedback.

6 TestExecution

- Testers: HR staff or QA team members
- Duration: Approximately 2 hours
- Reporting: Document pass/fail for each test case, including screenshots of CLI outputs and employees.csv contents.
- Defect Handling: Log any failures with steps to reproduce and expected vs. actual outcomes. Retest after fixes.

7 Assumptions

- Users have basic familiarity with command-line interfaces.
- Email sending is simulated (no actual SMTP server required).
- Testing starts with an empty or controlled employees.csv.

7. Problem Management (PM)

Problem management principles throughout lifecycle

The Problem Management lifecycle, typically includes stages like Identification, Logging, Categorization, Prioritization, Investigation (RCA), Resolution and Recovery, and Closure. The following core principles are applied throughout this lifecycle:

1. **Root Cause Focus:**
    - **Lifecycle Application:** This is central to the **Investigation** stage, where techniques like 5 Whys, Fishbone, and Fault Tree Analysis are used to find the underlying cause, not just treat symptoms. It also informs **Resolution and Recovery** by ensuring fixes are permanent. During **Problem Closure**, confirming the root cause is addressed is key.
    - **Goal:** Prevent problem recurrence by identifying and addressing the fundamental issues.
2. **Proactive and Reactive Approach:**
    - **Lifecycle Application:**
        - **Reactive:** Triggered by incidents. This involves **Problem Identification** (from recurring incidents), **Logging**, **Categorization**, **Prioritization**, **Investigation**, and **Resolution** of existing problems.
        - **Proactive:** Involves **Problem Identification** through trend analysis, system monitoring, and risk assessments before incidents occur. The lifecycle then follows similar steps to investigate and resolve potential problems.
    - **Goal:** Minimize the number and impact of incidents. Reactive PM fixes current issues; Proactive PM prevents future ones.
3. **Continuous Improvement:**
    - **Lifecycle Application:** This principle is woven throughout the entire lifecycle. Feedback from **Problem Closure** and post-resolution monitoring (**Verification and Monitoring**) feeds back into **Problem Identification** (e.g., identifying new trends) and improves the PM process itself (e.g., refining RCA techniques, updating knowledge bases). Post-Incident Reviews are crucial.
    - **Goal:** Enhance the efficiency and effectiveness of the problem management process over time, adapting to new technologies and organizational needs.
4. **Collaboration and Communication:**
    - **Lifecycle Application:** Essential at all stages.
        - **Identification & Logging:** Involves users, service desk, and monitoring tools.
        - **Investigation:** Requires input from various technical teams (developers, network, infrastructure).
        - **Solution Development & Resolution:** Needs coordination for implementing changes.
        - **Closure:** Stakeholders must be informed.
    - **Goal:** Ensure all relevant parties are involved, informed, and working together for comprehensive problem resolution and knowledge sharing.
5. **Documentation and Knowledge Sharing:**

- **Lifecycle Application:** Critical at every step.
  - **Logging:** Detailed recording of problem symptoms, impact, etc.
  - **Investigation:** Documenting RCA steps and findings.
  - **Resolution:** Recording solution details and implementation steps.
  - **Closure:** Comprehensive documentation of the entire process, updating the Known Error Database (KEDB).
- **Goal:** Build organizational knowledge, enable faster resolution of future similar problems, support training, and facilitate continuous improvement.

Tools used for PM across its lifecycle

- **1. Problem Identification & Logging:**
  - **Monitoring Systems:** Splunk, ELK Stack, Nagios, SolarWinds (for automated detection of anomalies and recurring errors).
  - **Issue Tracking/ITSM Tools:** JIRA, Bugzilla, ServiceNow (for logging incidents that may become problems, and for logging problems themselves).
  - **User Feedback Channels:** (Implicit) Email, helpdesk systems.
- **2. Problem Categorization:**
  - **ITSM Tools:** JIRA, ServiceNow (often have features to categorize problems by severity, type, impact area).
- **3. Problem Prioritization:**
  - **ITSM/Workflow Tools:** JIRA, Trello (can be used with priority matrices to manage and visualize problem priorities).
- **4. Problem Investigation (RCA):**
  - **Diagramming Tools (for Fishbone, Fault Tree):** Lucidchart, Draw.io, Microsoft Visio.
  - **RCA Methodologies (conceptual tools):** 5 Whys, Fishbone Diagram, Fault Tree Analysis, Pareto Analysis, Kepner-Tregoe.
  - **Log Analysis Tools:** Splunk, ELK Stack (to dive deeper into system logs for clues).
  - **Data Analytics Tools:** (Generic) Tools for analyzing incident data to identify patterns and trends.
- **5. Solution Development & Implementation:**
  - **Version Control Systems:** Git (for tracking code changes related to fixes).
  - **CI/CD Pipeline Tools:** Jenkins, GitLab CI (for automated testing and deployment of fixes).
  - **Change Management Systems:** (Often part of ITSM tools like ServiceNow) To manage the process of implementing solutions.
- **6. Verification and Monitoring (Post-Resolution):**
  - **Unit Testing Tools:** pytest, unittest.
  - **Integration Testing Tools:** Selenium, Postman.
  - **Load Testing Tools:** Apache JMeter.
  - **Automated Testing Suites:** For regression testing.
  - **Monitoring Systems:** (As in identification) Splunk, ELK Stack, Nagios, SolarWinds (to ensure stability and that the problem doesn't recur).
- **7. Problem Closure & Knowledge Sharing:**

    - **Knowledge Management Systems:** Known Error Database (KEDB), Confluence, SharePoint.
    - **ITSM Tools:** JIRA, ServiceNow (for formally closing problem records and linking to knowledge articles).

Best practices and industry standards in documentation

- **General Best Practices & Standards:**
    - **ITIL (Information Technology Infrastructure Library):** A widely adopted framework providing comprehensive guidance on IT service management, including detailed processes for Problem Management and the importance of documentation like Problem Records and KEDBs.
    - **ISO/IEC 20000:** An international standard for IT service management systems, which emphasizes the need for documented procedures and records for processes like problem management.
    - **Consistent Templates:** Using standardized templates for problem records, KEDB entries, and RCA reports ensures all necessary information is captured uniformly, making it easier to understand and analyze.
    - **Comprehensive Logs:** Detailed logging of all problem-related information: symptoms, impact, diagnostic steps, actions taken, individuals involved, timestamps, and resolutions.
    - **Knowledge Base Updates (KEDB):** Regularly updating the KEDB with details of new problems, their root causes, workarounds, and permanent solutions. This is a cornerstone of effective knowledge sharing.
    - **Clear Communication:** Ensuring documentation is clear, concise, and easily accessible to all stakeholders. This includes reports on root causes and solutions.
- **Documentation in Different Stages of the Problem Management Lifecycle:**
    - **1. Identification & Logging:**
        - **Practice:** Ensure accurate and detailed logging of the initial problem report.
        - **Documentation:** Problem ticket/record including:
            - Date and time of identification.
            - Source of identification (e.g., incident ID, monitoring alert).
            - Detailed description of symptoms.
            - Frequency of occurrence.
            - Impacted users, services, or systems.
            - Initial diagnostic steps taken (if any).
    - **2. Categorization & Prioritization:**
        - **Practice:** Document the assigned category (e.g., hardware, software, network) and priority (e.g., critical, high, medium, low) with justification.
        - **Documentation:** Update the problem record with:
            - Severity, Impact, Urgency assessment.
            - Problem Type (e.g., bug, configuration error).
            - Impact Area (e.g., functional, performance).
            - Calculated Priority Level.
    - **3. Investigation (RCA):**
        - **Practice:** Document the entire RCA process, including hypotheses,

tests, and findings.
    - **Documentation:**
        - RCA technique(s) used (e.g., 5 Whys transcript, Fishbone diagram).
        - Data collected and analyzed.
        - Identified root cause(s) with supporting evidence.
        - Failed hypotheses or paths explored.
- **4. Solution Development & Resolution:**
    - **Practice:** Document the proposed solution, any workarounds, and the plan for permanent resolution.
    - **Documentation:**
        - **Workaround details:** Steps to implement, effectiveness (if applicable).
        - **Permanent solution:** Description of the fix (e.g., code changes, configuration updates, hardware replacement).
        - **Implementation plan:** Steps, resources, timeline, rollback plan.
        - Test cases developed for the fix.
- **5. Verification and Monitoring:**
    - **Practice:** Document the results of testing and monitoring post-implementation.
    - **Documentation:**
        - Test results (unit, integration, regression).
        - Monitoring data confirming stability and non-recurrence.
        - User feedback post-fix.
- **6. Problem Closure:**
    - **Practice:** Obtain confirmation from relevant teams and document closure details comprehensively.
    - **Documentation:**
        - Final confirmation that the problem is resolved.
        - Summary of the problem, root cause, and solution.
        - Date of closure.
        - Update to the KEDB with all relevant information (Error Description, Root Cause, Error Status: Resolved, Permanent Fix, Implementation Plan, Resolution Status: Completed, Impact, Priority). (Deck 2, Slides 34-35)
        - Lessons learned.
- **7. Documentation and Knowledge Sharing (Ongoing):**
    - **Practice:** Maintain a detailed and accessible KEDB. Share lessons learned with relevant teams.
    - **Documentation:** KEDB entries, post-mortem reports, updated operational procedures, training materials.

8. Problem Categorisation and Prioritisation

Categories used for categorisation

1. **Severity:** The degree of impact the error has on the system or its testing.

- **Critical:** Prevents major functionality from being tested or indicates a system-breaking issue. Blocks further testing of a module.
- **Major:** Causes significant deviation from expected outcomes, impacts a key feature, or leads to incorrect results that could mislead development/users. May allow some other tests to proceed.
- **Minor:** A less significant issue, such as an incorrect error message format (if the core functionality is still testable) or a cosmetic issue in test output. Does not block other tests.

2. **Type:** The nature of the error.
    - **Code Logic Error:** A flaw in the program's algorithm or business logic (e.g., incorrect calculation, wrong conditional path).
    - **Data Handling Error:** Issues with how data is processed, stored, or retrieved (e.g., incorrect data type conversion, issues with file I/O).
    - **Configuration Error:** Problem related to setup, environment, or external dependencies (less common for these unit test errors but possible).
    - **Test Script Error:** The unit test itself is flawed (e.g., incorrect assertion, bad mock setup).
    - **Interface Mismatch:** Discrepancy between how a function/method is called and how it's defined, or how components interact.
3. **Impact Area (for the application, inferred from test failure):**
    - **Functional:** Affects a specific function or feature of the application (e.g., employee creation, data saving, email sending).
    - **Data Integrity:** Could lead to corrupted or incorrect data being stored or processed.
    - **Usability/User Experience (UX):** (Less direct from these unit tests) e.g., misleading error messages to the user.
    - **Testability:** The error makes it difficult to reliably test a part of the system.

Matrices for prioritisation

1. **Impact (on Development/Testing Process):**
    - **High:** Blocks significant testing, indicates a critical flaw in core functionality, or affects multiple modules/tests. Requires immediate attention.
    - **Medium:** Affects a specific feature's testing, leads to unreliable test results for a component, or indicates a notable bug. Needs prompt attention.
    - **Low:** Minor issue, workaround possible for testing, or affects non-critical test aspects. Can be addressed when time permits.
2. **Urgency (to fix for continued development/testing):**
    - **High:** Immediate fix needed to unblock testing or development, or to prevent propagation of errors.
    - **Medium:** Needs to be fixed in the near term to ensure test reliability or feature correctness.
    - **Low:** Can be scheduled for a later iteration.
3. **Frequency (if this type of error were to occur in production, or how often it blocks testing):**
    - **High:** Likely to occur often or affects a commonly used testing path.

    - **Medium:** Occurs under specific conditions or affects a moderately used testing path.
    - **Low:** Rare occurrence or affects an edge-case testing scenario.

Priority Level (derived from Impact, Urgency, and sometimes Frequency):

- **Critical:** (High Impact, High Urgency)
- **High:** (High Impact, Medium Urgency OR Medium Impact, High Urgency)
- **Medium:** (Medium Impact, Medium Urgency OR High Impact, Low Urgency OR Low Impact, High Urgency)
- **Low:** (Low Impact, Low Urgency OR Medium Impact, Low Urgency OR Low Impact, Medium Urgency)

Category & priority level for the error encountered

Here are the 4 errors from my unit tests:

Error 1: FAIL: test_employee_from_dict_invalid_salary

- **Description:** Assertion error due to mismatched print message for invalid salary in Employee.from_dict.
- **Categorization:**
    - **Severity:** Minor (The core logic of from_dict returning None for invalid salary is likely correct, but the test's expectation of the error message is outdated. Doesn't block testing of from_dict's main behavior).
    - **Type:** Test Script Error (The test assertion needs updating to match the actual program output).
    - **Impact Area:** Testability (Specifically, the accuracy of this particular test assertion).
- **Prioritization:**
    - **Impact (Testing):** Low (The functionality itself isn't broken, just the test's check of a specific string).
    - **Urgency (Fix):** Medium (Good to keep tests accurate, but doesn't block other development).
    - **Frequency (Blocking):** Low (Only this specific assertion is affected).
    - **Priority Level: Low**

Error 2: FAIL: test_save_employees

- **Description:** Assertion error because a specific write() call for the second employee's data was not found.
- **Categorization:**
    - **Severity:** Major (If _save_employees isn't saving all data correctly, or if to_dict has an issue, this is a significant problem for data persistence).
    - **Type:** Could be Data Handling Error (in _save_employees or to_dict) OR Test Script Error (if test data setup is inconsistent with to_dict output). (Further RCA will clarify).
    - **Impact Area:** Data Integrity, Functional (Saving employee data).
- **Prioritization:**
    - **Impact (Testing):** High (Core functionality of saving data is in question).
    - **Urgency (Fix):** High (Data persistence is critical).
    - **Frequency (Blocking):** High (Affects any test relying on correct data saving).
    - **Priority Level: Critical**

Error 3: FAIL: test_send_email_auth_error

- **Description:** Assertion error because the expected print message for an SMTP authentication error was not found.
- **Categorization:**
    - **Severity:** Major (Indicates that error handling for email authentication might not be working as expected, or an unexpected error like NameError is occurring first).
    - **Type:** Code Logic Error (Likely an issue in _send_confirmation_email preventing the SMTPAuthenticationError from being caught and handled as expected, or a preceding error).
    - **Impact Area:** Functional (Email sending error handling), Testability.
- **Prioritization:**
    - **Impact (Testing):** Medium (Affects testing of a specific error path in email functionality).
    - **Urgency (Fix):** High (Correct error handling is important).
    - **Frequency (Blocking):** Medium (Blocks testing this specific error scenario).
    - **Priority Level: High**

Error 4: FAIL: test_send_email_success

- **Description:** Assertion error because the expected print message for a simulated successful email send was not found.
- **Categorization:**
    - **Severity:** Minor to Major (Depends on whether the issue is just the print message or if the email sending logic itself has a flaw. Given Error 3, it's likely related to issues within _send_confirmation_email).
    - **Type:** Code Logic Error (in _send_confirmation_email regarding what's printed) OR Test Script Error (test expecting a message that the code isn't designed to give in that path). (Further RCA will clarify).
    - **Impact Area:** Functional (Email sending confirmation/simulation), Testability.
- **Prioritization:**
    - **Impact (Testing):** Medium (Affects testing of the email success path).
    - **Urgency (Fix):** Medium (Important to verify email functionality, even if simulated).
    - **Frequency (Blocking):** Medium (Blocks testing this success scenario).
    - **Priority Level: Medium**

9. Root Cause Analysis (RCA) and Resolution

Technique used for RCA

Steps taken in identifying the root cause of error encountered

Root cause of the error

10. Resolution

Steps taken to fix the error

Rerunning test case

KEDB

- **Error 1: FAIL: test_employee_from_dict_invalid_salary**

- **Error Description (from traceback):**

```
    AssertionError: expected call not found.
Expected: print("Error creating employee from dictionary data: could not
convert string to float: 'not-a-number'. Data: {...}")
Actual: print("Error creating employee: Invalid salary format 'not-a-
number'. Data: {...}")
```

- **RCA Technique Used:** Code Inspection and Comparison.
- **Steps to Find Root Cause:**
    1. Compared the Expected print message in the test error with the Actual print message.
    2. The difference is in the initial part of the string:
        - Expected: "Error creating employee from dictionary data: could not convert string to float: ..."
        - Actual: "Error creating employee: Invalid salary format ..."
    3. Inspected the Employee.from_dict method in employee_management.py:

```
    # employee_management.py
@staticmethod
def from_dict(data_dict):
    try:
        salary_str = data_dict.get("Salary", "0")
        try:
            salary_float = float(salary_str)
        except ValueError:
            # Handle only the float conversion error here
            print(f"Error creating employee: Invalid salary format
'{salary_str}'. Data: {data_dict}") # THIS IS THE ACTUAL PRINT
            return None
        # ...
        return Employee(...)
    except ValueError as e: # Catches validation errors from
Employee.__init__
        print(f"Error creating employee object from dictionary: {e}. Data:
{data_dict}") # This was likely the old print the test expected
        raise e
    # ...
```

-

    1. The code explicitly prints f"Error creating employee: Invalid salary format '{salary_str}'. Data: {data_dict}" when the float(salary_str) conversion fails.
    2. The test case test_employee_from_dict_invalid_salary in test_employee_system.py has an outdated assertion for this specific print message.

- **Resolution (Code Update):**
  Update the assertion in test_employee_system.py to match the actual print statement from the employee_management.py code.

File: test_employee_system.py
Method: test_employee_from_dict_invalid_salary

```
    # ...
```

```
def test_employee_from_dict_invalid_salary(self):
    invalid_data = VALID_EMP_DATA_1.copy()
    invalid_data["Salary"] = "not-a-number"
    with patch('builtins.print') as mock_print:
        emp = Employee.from_dict(invalid_data)
        self.assertIsNone(emp) # Expecting None based on current
implementation
        # Check the specific error message printed by from_dict
        # OLD:
        # mock_print.assert_called_once_with(f"Error creating employee from
dictionary data: could not convert string to float: 'not-a-number'. Data:
{invalid_data}")
        # NEW (FIXED):
        mock_print.assert_called_once_with(f"Error creating employee:
Invalid salary format 'not-a-number'. Data: {invalid_data}")
```

- **Rerun Test Case:**
  After applying this change to test_employee_system.py, the test_employee_from_dict_invalid_salary test case is expected to pass.
- **Record the Problem in a Known Error Database (KEDB):**

Field	Details
1. Known Error Record	
Error ID	KEDB-001
Error Description	Unit test test_employee_from_dict_invalid_salary fails due to mismatched expected error message.
Symptoms	AssertionError in test output, actual print message differs from expected.
Root Cause	The unit test assertion for the error message printed by Employee.from_dict upon invalid salary conversion was not updated after the program's error message was changed to be more specific.
Error Status	Resolved
Date/Time Identified	[Current Date/Time]
2. Workarounds	
Temporary Solutions	N/A (Direct fix preferred)
3.	

Resolution

• Permanent Fix	• Updated the mock_print.assert_called_once_with in test_employee_from_dict_invalid_salary to match the actual, more specific error message generated by the Employee.from_dict method.
• Implementation Plan	• Modify test_employee_system.py as described. Re-run unit tests to confirm fix.
• Resolution Status	• Completed
• **4. Impact and Priority**	
• Impact Assessment	• Low impact on core functionality; primarily affected test script accuracy.
• Priority Level	• Low
• **7. Ownership**	
• Owner	• Junior Programmer

- **Error 2: FAIL: test_save_employees**
- **Error Description (from traceback):**

```
    AssertionError: write('BF002,Bob
Johnson,Marketing,60000,bob.j@bitfutura.test,987-654-3210\n') call not
found
```

- **RCA Technique Used:** Code Inspection, Data Tracing.
- **Steps to Find Root Cause:**
  1. The error indicates that the expected string for the second employee (VALID_EMP_DATA_2) was not written to the mock file.
  2. The test setup for test_save_employees:

```
emp1 = Employee.from_dict(VALID_EMP_DATA_1)
emp2 = Employee.from_dict(VALID_EMP_DATA_2)
self.ems._employees_list = [emp1, emp2]
self.ems._save_employees()
# ...
expected_line2 = DELIMITER.join(VALID_EMP_DATA_2.values()) + '\n'
handle.write.assert_any_call(expected_line2)
```

-
    1. VALID_EMP_DATA_2 is defined in test_employee_system.py as:

```
      VALID_EMP_DATA_2 = {
    "Employee ID": "BF002", "Name": "Bob Johnson", "Department":
"Marketing",
    "Salary": "60000", # Note: Salary is "60000"
    "Email": "bob.j@bitfutura.test", "Contact Details": "987-654-3210"
}
```

So, expected_line2 will contain "60000" for the salary.

2. Inspect the Employee.to_dict() method in employee_management.py, which is called by _save_employees:

```
      # employee_management.py
def to_dict(self):
    return {
        # ...
        "Salary": f"{self.salary:.2f}", # Salary is formatted to 2 decimal
places
        # ...
    }
```

When emp2 (created from VALID_EMP_DATA_2) calls to_dict(), its salary (which is 60000.0 as a float internally) will be formatted as "60000.00".

3. The _save_employees method writes this formatted salary.
4. Therefore, the actual line written to the file for emp2 will contain "60000.00" for salary, but the test expects "60000". This mismatch causes the assertion to fail.

- **Resolution (Code Update):**
  Update the VALID_EMP_DATA_2 dictionary in test_employee_system.py to have its "Salary" field formatted to two decimal places, consistent with the output of the Employee.to_dict() method.

File: test_employee_system.py
Test Data Definition:

```
      # ...
VALID_EMP_DATA_2 = {
    "Employee ID": "BF002", "Name": "Bob Johnson", "Department":
"Marketing",
    # OLD: "Salary": "60000",
    "Salary": "60000.00", # NEW (FIXED): Ensure consistency with to_dict()
output
    "Email": "bob.j@bitfutura.test", "Contact Details": "987-654-3210"
}
# ...
```

- **Rerun Test Case:**
  After applying this change to test_employee_system.py, the test_save_employees

test case is expected to pass.

- **Record the Problem in a Known Error Database (KEDB):**

Field	Details
1. Known Error Record	
Error ID	KEDB-002
Error Description	Unit test test_save_employees fails because the expected written string for an employee's salary does not match the actual formatted salary string.
Symptoms	AssertionError in test output: a specific write() call with unformatted salary is not found.
Root Cause	The test data (VALID_EMP_DATA_2) used an unformatted salary string ("60000"), while the Employee.to_dict() method formats salaries to two decimal places ("60000.00") before saving. The test assertion expected the unformatted version.
Error Status	Resolved
Date/Time Identified	[Current Date/Time]
2. Workarounds	
Temporary Solutions	N/A (Direct fix preferred)
3. Resolution	
Permanent Fix	Updated the "Salary" value in the VALID_EMP_DATA_2 test dictionary in test_employee_system.py from "60000" to "60000.00" to match the output format of Employee.to_dict().
Implementation Plan	Modify test_employee_system.py as described. Re-run unit tests to confirm fix.
Resolution Status	Completed
4. Impact and Priority	
Impact	High impact on testing data persistence. If unaddressed, it masks

Assessment	whether saving is truly correct.
• Priority Level	• Critical
• **7. Ownership**	
• Owner	• Junior Programmer

•

- **Error 3: FAIL: test_send_email_auth_error**
- **Error Description (from traceback):**

```
    AssertionError: print('Email Error: Authentication failed. Check
username/password.') call not found
```

IGNORE_WHEN_COPYING_START
content_copy download
Use code with caution.
IGNORE_WHEN_COPYING_END

- **RCA Technique Used:** Code Inspection, Debugging (Mental Walkthrough).
- **Steps to Find Root Cause:**
    1. The test test_send_email_auth_error mocks smtplib.SMTP such that its login method raises smtplib.SMTPAuthenticationError.
    2. The test expects the except smtplib.SMTPAuthenticationError: block in _send_confirmation_email to be triggered, which should print "Email Error: Authentication failed. Check username/password.".
    3. This message is not being printed. This suggests either the SMTPAuthenticationError is not being raised as expected by the mock, or another exception is occurring before the smtp.login() call.
    4. Inspecting _send_confirmation_email in employee_management.py:

```
      # employee_management.py
def _send_confirmation_email(self, employee_email, employee_name):
    # ...
    try:
        # ...
        msg = EmailMessage()
        msg['Subject'] = subject # Potential NameError: 'subject' is not
defined
        msg['From'] = EMAIL_ADDRESS
        msg['To'] = employee_email
        msg.set_content(body)   # Potential NameError: 'body' is not
defined

        with smtplib.SMTP(SMTP_SERVER, SMTP_PORT) as smtp:
            smtp.starttls()
            smtp.login(EMAIL_ADDRESS, EMAIL_PASSWORD) # This is where
SMTPAuthenticationError is expected
            smtp.send_message(msg)
        # ...
    except smtplib.SMTPAuthenticationError:
        print("Email Error: Authentication failed. Check
username/password.")
    # ...
    except Exception as e: # This generic handler will catch NameError
        print(f"An error occurred while sending the email: {e}")
```

IGNORE_WHEN_COPYING_START
content_copy download
Use code with caution. Python
IGNORE_WHEN_COPYING_END

5. The variables subject and body are used before they are assigned a value within the _send_confirmation_email method. This will raise a NameError.
6. The NameError occurs before smtp.login() is called.
7. The generic except Exception as e: block catches this NameError, printing f"An error occurred while sending the email: {e}" (where e would be the NameError).
8. Therefore, the specific except smtplib.SMTPAuthenticationError: block is never reached.

- **Resolution (Code Update):**
  Define subject and body variables at the beginning of the _send_confirmation_email method in employee_management.py.

File: employee_management.py
Method: _send_confirmation_email

```
      # ...
def _send_confirmation_email(self, employee_email, employee_name):
    # FIX: Define subject and body
    subject = f"Welcome to BitFutura, {employee_name}!" # Example subject
    body = f"Dear {employee_name},\n\nYour employee record has been
successfully processed.\n\nRegards,\nBitFutura HR" # Example body

    try:
        EMAIL_ADDRESS = os.environ.get('EMAIL_USER')
        EMAIL_PASSWORD = os.environ.get('EMAIL_PASS')
        SMTP_SERVER = 'smtp.example.com' # Replace with your SMTP server
        SMTP_PORT = 587 # Standard TLS port

        if not EMAIL_ADDRESS or not EMAIL_PASSWORD:
            print("Email credentials not configured. Skipping actual email
sending.")
            # Optional: Add a simulation print here if desired for this
path
            # print(f"SIMULATION: Email to {employee_email}, Subject:
{subject}")
            return

        msg = EmailMessage()
        msg['Subject'] = subject # Now defined
        msg['From'] = EMAIL_ADDRESS
        msg['To'] = employee_email
        msg.set_content(body) # Now defined

        with smtplib.SMTP(SMTP_SERVER, SMTP_PORT) as smtp:
            smtp.starttls()
            smtp.login(EMAIL_ADDRESS, EMAIL_PASSWORD) # This should now be
reached
            smtp.send_message(msg)
        print("Actual confirmation email sent successfully.")

    except smtplib.SMTPAuthenticationError:
        print("Email Error: Authentication failed. Check
username/password.")
    except smtplib.SMTPConnectError:
        print(f"Email Error: Could not connect to SMTP server
{SMTP_SERVER}:{SMTP_PORT}.")
    except Exception as e:
        print(f"An error occurred while sending the email: {e}")
# ...
```

IGNORE_WHEN_COPYING_START
content_copy download
Use code with caution. Python
IGNORE_WHEN_COPYING_END

- **Rerun Test Case:**
  After applying this change to employee_management.py, the test_send_email_auth_error test case is expected to pass.
- **Record the Problem in a Known Error Database (KEDB):**

Field	Details
1. Known Error Record	
Error ID	KEDB-003
Error Description	Unit test test_send_email_auth_error fails to detect the expected print message for SMTP authentication failure.
Symptoms	AssertionError in test output; expected specific auth error message not printed. Instead, a generic error message related to a NameError was likely printed (though not explicitly asserted by this test).
Root Cause	NameError occurred in _send_confirmation_email due to undefined subject and body variables. This NameError was caught by a generic except Exception block before the code could reach smtp.login() to trigger the mocked SMTPAuthenticationError.
Error Status	Resolved
Date/Time Identified	[Current Date/Time]
2.	

Workarounds	
Temporary Solutions	N/A (Direct fix preferred)
3. Resolution	
Permanent Fix	Defined subject and body variables at the beginning of the _send_confirmation_email method in employee_management.py.
Implementation Plan	Modify employee_management.py as described. Re-run unit tests to confirm fix.
Resolution Status	Completed
4. Impact and Priority	
Impact Assessment	High impact on testing email error handling. Prevents verification of correct behavior for authentication failures. Could mask actual email sending issues in production if NameError persisted.
Priority Level	High
7. Ownership	
Owner	Junior Programmer

•

- ## Error 4: FAIL: test_send_email_success
- **Error Description (from traceback):**

```
    AssertionError: print("Simulating email send to recipient@example.com
with subject 'Employee Action Confirmation'") call not found
```

IGNORE_WHEN_COPYING_START
content_copy download
Use code with caution.
IGNORE_WHEN_COPYING_END

- **RCA Technique Used:** Code Inspection, Test Logic Review.
- **Steps to Find Root Cause:**
  1. The test test_send_email_success expects a specific simulation print message.
  2. The _send_confirmation_email method (after the fix for Error 3) is

structured as follows:

- If EMAIL_ADDRESS or EMAIL_PASSWORD are not found (via os.environ.get), it prints "Email credentials not configured. Skipping actual email sending." and returns.
- If credentials are found, it proceeds to set up EmailMessage and attempts the SMTP operations. Upon successful (mocked) completion of smtp.send_message(msg), it prints "Actual confirmation email sent successfully.".

3. The test test_send_email_success mocks os.environ.get to return 'dummy_value' for credentials, meaning the "credentials not configured" path is skipped.
4. Therefore, the code path taken leads to the "Actual confirmation email sent successfully." print, not the "Simulating email send..." print that the test asserts.
5. Additionally, the test asserts self.assertEqual(mock_msg_instance['Subject'], 'Employee Action Confirmation'). However, the fix for Error 3 sets the subject dynamically: subject = f"Welcome to BitFutura, {employee_name}!". This will also cause an assertion failure for the subject line if not addressed.

- **Resolution (Code Update):**
  Update the assertions in test_send_email_success in test_employee_system.py to match what the _send_confirmation_email method actually does and prints when credentials are provided and the (mocked) send is successful. This involves changing the expected print message and the expected subject line.

File: test_employee_system.py
Method: test_send_email_success

```
      # ...
@patch('employee_management.os.environ.get')
@patch('employee_management.smtplib.SMTP')
@patch('employee_management.EmailMessage')
@patch('builtins.print')
def test_send_email_success(self, mock_print, mock_email_msg_cls,
mock_smtp_cls, mock_env_get):
    mock_env_get.side_effect = lambda key: 'dummy_value' if key in
('EMAIL_USER', 'EMAIL_PASS') else None
    mock_smtp_instance = MagicMock()
    mock_smtp_cls.return_value.__enter__.return_value = mock_smtp_instance
    mock_msg_instance = MagicMock()
    mock_email_msg_cls.return_value = mock_msg_instance

    test_email = "recipient@example.com"
    test_name = "Recipient Name"
    self.ems._send_confirmation_email(test_email, test_name) # Assumes
Error 3 fix is applied to employee_management.py

    # Check EmailMessage content
    mock_email_msg_cls.assert_called_once()
    self.assertEqual(mock_msg_instance['To'], test_email)
    self.assertEqual(mock_msg_instance['From'], 'dummy_value') # From
EMAIL_USER

    # FIX: Align subject with what the code now produces (after Error 3
fix)
```

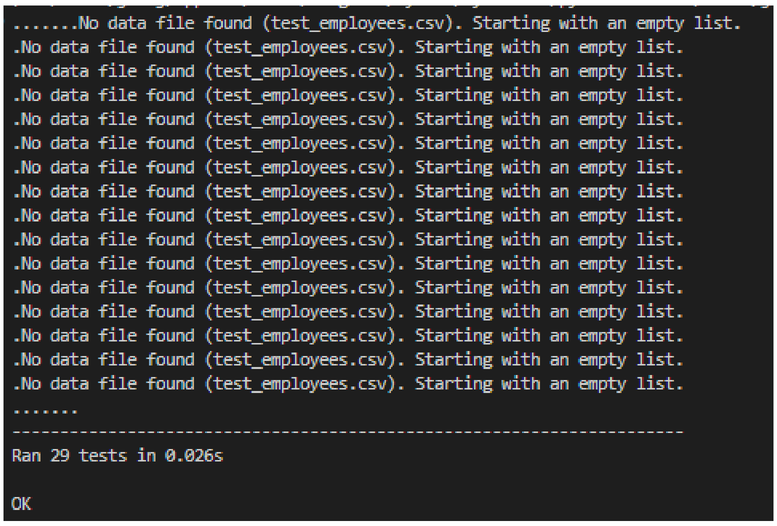
```
    expected_subject = f"Welcome to BitFutura, {test_name}!"
    self.assertEqual(mock_msg_instance['Subject'], expected_subject)

    # FIX: Align print assertion with what the code now prints on (mocked)
success
    mock_print.assert_any_call("Actual confirmation email sent
successfully.")
    # Remove or comment out the old simulation print assertion if it's no
longer relevant for this test's intent:
    # mock_print.assert_any_call(f"Simulating email send to {test_email}
with subject 'Employee Action Confirmation'")

    # Optional: Verify SMTP calls if testing the actual send path more
deeply
    mock_smtp_cls.assert_called_with('smtp.example.com', 587)
    mock_smtp_instance.starttls.assert_called_once()
    mock_smtp_instance.login.assert_called_once_with('dummy_value',
'dummy_value')

mock_smtp_instance.send_message.assert_called_once_with(mock_msg_instance)
# ...
```

IGNORE_WHEN_COPYING_START
content_copy download
Use code with caution. Python
IGNORE_WHEN_COPYING_END

- **Rerun Test Case:**
  After applying the fix for Error 3 to employee_management.py and these changes to test_send_email_success in test_employee_system.py, this test case is expected to pass.
- **Record the Problem in a Known Error Database (KEDB):**

Field	Details
1. Known Error Record	
Error ID	KEDB-004
Error Description	Unit test test_send_email_success fails due to mismatched expected print message and email subject.
Symptoms	AssertionError in test output; test expects a "Simulating email send..." message and a fixed subject, but code prints "Actual confirmation..." and uses a dynamic subject when (mock) credentials are provided.
Root Cause	The test test_send_email_success was asserting a simulation message and a static subject, but it configured mocks such that the code would attempt an "actual" (mocked) send. The program logic for this path prints a different success message and uses a dynamic subject. The NameError (from KEDB-003) also masked this.

Error Status	Resolved
Date/Time Identified	[Current Date/Time]
2. Workarounds	
Temporary Solutions	N/A (Direct fix preferred)
3. Resolution	
Permanent Fix	1. Applied fix for KEDB-003 (defined subject and body in _send_confirmation_email). 2. Updated assertions in test_send_email_success to expect the "Actual confirmation email sent successfully." print message and the dynamically generated subject line that _send_confirmation_email produces when (mocked) credentials are present.
Implementation Plan	Modify employee_management.py (for KEDB-003 fix) and test_employee_system.py (for this test's assertions). Re-run unit tests.
Resolution Status	Completed
4. Impact and Priority	
Impact Assessment	Medium impact on testing the success path of email sending.
Priority Level	Medium
7. Ownership	
Owner	Junior Programmer



```
.......No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.No data file found (test_employees.csv). Starting with an empty list.
.......
----------------------------------------------------------------------
Ran 29 tests in 0.026s

OK
```