# Task 2 – Planning and Designing a Procedural Programming Application

*Scenario: You work for a software development company, Financial IT Solutions Ltd, who have been recently asked by a client to develop a payroll processing application.*

*The documentation provided by the client does not contain enough information about the application requirements. You have been asked to interview the client to gather further information about the application requirements and produce a simple design document. The design document should clearly outline the required inputs, processing taking place, and desired outputs.*

To develop a comprehensive design document for the payroll processing application as requested by the client, and based on the scenario provided, we will need to expand on the initial assumptions and work breakdown structure to include additional considerations. This enhancement will ensure a robust and user-friendly application that aligns with the client's needs, even if not explicitly stated. Following the established conversational and straightforward style, let's outline the missing points and integrate them into the existing framework.

Based on the fact that our client didn’t provide enough information about the application requirements, some assumptions were made:

1. **Convention:** We will be following snake\_convention as it’s very common and readable.
2. **Database Structure:** Assumed that the client requested a CSV database structure as it's easy to edit for non-technical users, each CSV file will represent a table.
3. **Tax calculation:** Assumed the taxes are the client’s responsibility
4. **Generate Preset of Data:** Assuming that the client requested the data generation from the program and wrote it to separate CSV files, we created the flag for this assumption, that we can receive as the user input in future versions of the application.
5. **Reading data from the database:** Assuming that our client wants to read data from CSV files if changes were made manually
6. **Calculation:** Following the client's requirements, we will be calculating the following:
   * + Based on the completed courses, calculate and display the Basic Pay, Bonus Pay, and Total Pay earned by each tutor
     + The overall Profit / Loss made by the company for these completed courses
7. **Statistic Display:** Assuming that the client requested calculated statistics to be displayed for each task
8. **Create Text:** Assumed that the text files with statistics were requested for easy accessibility, they also requested to be written separately from CSV files, in the ‘documentation’ directory.
9. **Update tables:** Assuming that the client requested statistics to be integrated into the database, we rewrite the files CSV with newly added columns: ‘tutor\_total\_earnings’ to ‘tutors.csv’ and ‘module\_total\_profit’ to ‘modules.csv’.

## Work Breakdown Structure:

We will be using a Work Breakdown Structure to represent our application functionality and justification for these approaches:

1. **Relative path:** Importing and receiving the path of the application file to create a relative path to the Database directory
2. **Optional generation of the tables:** Based on our assumptions generate tables and write them into the CSV files.
   1. For the data storage in the scope of the application, we will be using LISTS as it is easy to read and understand.
   2. We also going to create a dictionary that contains table headers, where the table name is a key and its value is a list of strings representing related headers
   3. Python flexibility allows us to use different types of data in the scope of lists or dictionaries.
   4. We also will create a ‘mapped\_data’ dictionary, which contains the key of a file name with (.csv) extension and as a value list of its data
   5. Additional function to write the CSV files, this function will be reused later in the code.
3. **Read files:** If the client does not want to generate tables, we read them and recreate data structures that follow our data structure.
4. **Datasets:** We will create 3 data sets, ‘modules\_data’, ‘tutors\_data’ and ‘completed\_courses\_data’, they will be lists of rows without headers. The headers we will store in the ‘table\_headers’ dictionary.
5. **Calculations:** When we have all the data we can proceed to calculations:
   1. The first function will represent Tutors Total Earnings, it will receive as parameters all 3 of our datasets and return the updated ‘tutors\_data’ dataset (list). The function iterates through 3 datasets and combines data following its connections
   2. The second calculation will be implemented to calculate the Module Total Profit and return the updated ‘modules\_data’ dataset (list).
   3. For easy error handling and to avoid errors in conversations we implemented one more function called ‘safe\_cast’, which receives value and format to what it should be converted, if the errors occur, it will return the default value (in the scope of function it’s set by default to None.
   4. After each calculation we add a new column to the headers to correctly store the data and write them if it was required.
   5. If errors occur during these calculations, the headers won’t be written as calculations aren’t done and the error will be displayed
6. **Display results:**
   1. To display results we will write the separate two functions that receive a table as a parameter. with prebuilt structure displays ‘tutors\_data’ and ‘modules\_data’
   2. In this function we iterate through the dataset and compile the string
   3. We will also return the built statistic as a string to be able to use it later
7. **Write statistics to text files:**
   1. To achieve that we will need to define another directory called ‘documentation’ and build a relative path to the statistic files
   2. We implement this task through one simple function that uses a string parameter as the content of the txt file and the file path with provided
   3. We combine those paths before in separate variables with predefined file names and pass them as parameters
   4. We also implement it via the WITH function and TRY function to ensure the catching and printing of possible errors
8. **Update CSV tables:**
   1. Combining data into ‘mapped\_data’ we will be able to reuse the same function that we use for generating the CSV files to write updated files.
   2. We also will pass updated ‘table\_headers’ and updated presents, the same file names will be used, and we will simply rewrite them.
   3. In the write\_csv\_file function we will ensure that the directory exists, otherwise create it without an error
   4. In this function we iterate through each line in each dataset and build file content, one long string with the new line characters
      1. This needs to be done because we will be able to write the file in one go to save some usage of the file and update them at once
   5. We also will use UTF-8 encoding during our write functions as it’s highly recommended in work with CSV files.

# Task 3 – Building a Procedural Programming Application

*Based on the requirements identified and design documentation produced in Task 2, you must now build the required application using the Python programming language and implement the procedural programming paradigm. Your code should adhere to coding conventions and standards.*

The application was built in Python and uses a relative path to execute and create necessary files.

##### The file structure of the application:

database\

completed\_courses.csv

modules.csv

tutors.csv

documentation\

compiled\_module\_stats.txt

compiled\_tutor\_stats.txt

ONAssignment\_Task2.py

* Built solution displays updated tables and re-write database files.A screenshot of a computer

  Description automatically generated

# Task 4 – Maintaining, Supporting, and Testing a Procedural Programming Application

*To conclude the project, you have been asked to produce a short report containing the following;*

1. *Tests conducted on the application*
2. *Recommendations for improvements*
3. *Supporting documentation*

## Tests Conducted on the Application

Each function within the application was tested individually for accuracy and reliability. This includes the data generation functions, reading and writing CSV files, calculation functions for tutor earnings and module profits, and error handling mechanisms. Mock data was used to ensure that calculations were correct and that data structures were properly managed.

##### Relative path

Error, in this case, represents that the path wasn’t relevant, we receive the path of our file using the build-in library ‘os’. In this case, our application is standalone and could run in any environment.

A screenshot of a computer

Description automatically generated

##### Error Handling and Edge Case Testing

Special attention was given to testing how the application handles errors (e.g., incorrect data types in CSV files) and edge cases (e.g., missing data for certain fields). This ensured the robustness of the application under less-than-ideal conditions we implemented the safe\_cast function that gives us a warning if the datatype was incorrect.

A screenshot of a computer

Description automatically generated

## Recommendations for Improvements

* **GUI Interface:** Introducing a graphical user interface (GUI) for non-technical users to interact with the application more intuitively. This could simplify operations such as data input, initiating calculations, and reviewing statistics through a reusable menu, that validates user input.
* **Database Integration:** While CSV files offer simplicity, a more robust database system (e.g., SQLite) could provide better data integrity, security, and scalability. This would be particularly useful as the volume of data grows.
* **Advanced Analytics:** Incorporate advanced analytics features to provide deeper insights, such as trend analysis over time for tutor earnings and module profitability. This could help in strategic planning and decision-making.
* **Security Measures:** Enhance security measures for data handling, especially if sensitive information like the personal details of tutors is involved. This could include encryption of CSV files and secure access controls or transfer database to SQLite.

## Supporting documentation

### Application Architecture

* The application follows a modular architecture, separating concerns into data handling, calculation logic, and displaying results in two formats, display and files.
* Python is used for processing the information, with a focus on readability and maintainability, using only the Procedural Programming paradigm.

### CSV File Structure

* **Tutors.csv:** Contains tutor details including Tutor ID, Tutor Name, Tutor’s Basic Daily Rate, Tutor’s Bonus Per Student and as a result – Tutor’s Total Earnings.
* **Modules.csv:** Lists modules, including Module ID, Module Name, Module Duration, Price Per Person for each Module and as a result – Total Profit for each Module.
* **Completed\_courses.csv:** Store records of completed courses, linking Tutors to Modules by their IDs and Number of Students.

### Calculation Logic

* **Tutor Earnings:** Calculated based on completed courses, applying predefined rates for basic and bonus pay.
* **Module Profits:** Calculated as the revenue generated from module enrolments minus the payments made to tutors, to determine overall profitability.

### Data Handling

* Data import and export utilize Python’s internal methods for reading and writing CSV files, with multiple error handling to manage inconsistencies or malformed data.

### Installation

* Begin by downloading the application package and as it uses a relative path, can be run from any directory as far as it’s fully extracted.