

s

TASCOMP

**Software Design Document**

ReportPro

|  |  |
| --- | --- |
|  |  |
|  |  |
| Reference: | SDD-TA-241025.0 |
| Revision: | 1 |
| Revision Date: | 11th November 2024 |
| Revision Author: | Tom Atkinson |

**Revision History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Revision** | **Revision Date** | **Revision Author** | **Reviewed by** | **Review Date** |
| 0 | 25th Oct 2024 | Tom Atkinson | Ashley Tizard | 6th Nov 2024 |
| **Description** | | | | |
| Initial version of document created | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Revision** | **Revision Date** | **Revision Author** | **Reviewed by** | **Review Date** |
| 1 | 11th Nov 2024 | Tom Atkinson |  |  |
| **Description** | | | | |
| Changes made based on feedback from initial review. | | | | |

**Contents**

[1. Introduction 1](#_Toc182816975)

[2. Overview 2](#_Toc182816976)

[3. Structure Diagram 3](#_Toc182816977)

[4. QuickStats Data 4](#_Toc182816978)

[5. Machine Learning Data Analysis 5](#_Toc182816979)

[5.1 Data Core 5](#_Toc182816980)

[5.2 AI Data Analysis 5](#_Toc182816981)

[6. Databases 8](#_Toc182816982)

[6.1 Output Database 8](#_Toc182816983)

[6.2 Forecast Database 9](#_Toc182816984)

[7. Report Configuration 11](#_Toc182816985)

[8. Report Generator 12](#_Toc182816986)

[8.1 Data Layer 13](#_Toc182816987)

[8.2 Generation Layer 14](#_Toc182816988)

[8.3 Output Layer 15](#_Toc182816989)

[9. Output Microservice 16](#_Toc182816990)

# Introduction

ReportPro is planned to be a tool used to generate a managerial PDF report every set period (quarterly by default.) The data behind these reports will be cleaned data based off the current QuickStats output databases.

Additionally, the reports will include forecasted data. This will be done using a machine learning AI model which is trained off the existing QuickStats output data. This will be developed using R, a popular coding language for data analysis and machine learning.

Resulting PDFs will be partially customisable. This will be done through a front-end configuration program where users can decide what information is to be included in the resulting PDF as well as some styling options.

# Overview

The entire ReportPro suite will be in three separate Git Repositories.

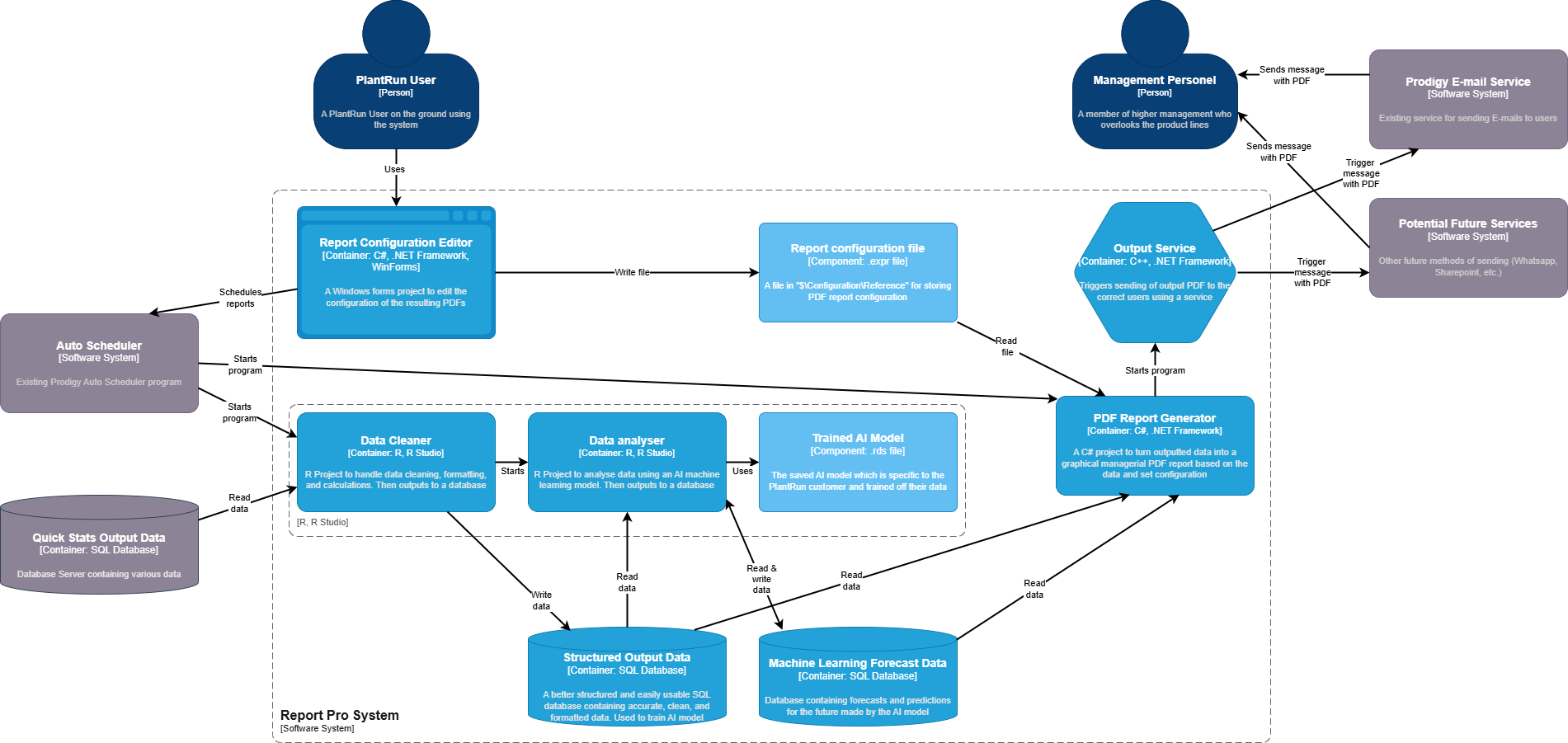
* R Studio Machine Learning and Data Analysis code.
* Report Configuration Editor.
* Report Generator.

The whole application will be a closed system. This means that no editing of the databases should be done from outside the ReportPro system.

There will also be a handshaking style system put in place to ensure no reports are taking place as the data is being edited.

# Structure Diagram

A level 2 C4 structure diagram for this project is shown below. Each required component has its own section in this document.



# QuickStats Data

The existing QuickStatsOutput data is in the form of SQL databases. These databases contain all the PlantRun downtime monitoring data for the selected period. It has the data formatted in many ways too:

* Monthly Based
* Weekly Based
* Daily Based
* Batch Based
* Operator Based
* Product Based
* Shift Based

This data will be the basis for everything involved in ReportPro. It will first be cleaned and formatted into a database, then analysis will be performed including calculations, finding trends, and AI model training. This will result in the data which is saved in the main SQL databases.

For ReportPro to be a success it is vital for the QuickStats Data it is using to be accurate and in the same format across all customer sites.

It is known that QuickStats data is not perfect. Another goal for this analysis application is to find flaws within the dataset itself. This will be found through the AI model but also issues may be picked up throughout the development process.

# Machine Learning Data Analysis

The R code responsible for the data manipulation will be split into two separate R projects, one for each of the respective databases.

## Data Core

This project will do the base processing of the data from QuickStats into a separate database, cleaning it in the process.

The code here will be fairly simple. Only basic analysis will take place and some simple value validation to check that the data is in a valid format. This is all done in preparation for the AI Data model to begin analysis.

It will keep all the cleaned QuickStats output data within a single database and update that database when appropriate.

This code will only ouput to the output database.

Depending on the quality of the QuickStats output data this program may be very small and simplistic.

## AI Data Analysis

This R project is used for the AI/machine learning aspect. It will review the outputted, clean data from the new output database and then begin to perform analysis.

It will start by loading the data from the database into R so it can read values from it.

**Model Development**

Next it will attempt to load the machine learning model if one already exists:

* If a saved model is available, the project will load it using readRDS(). This allows the project to build upon previous data.
* If no model is found, the project will proceed to create a new model from scratch. The project will incorporate a loop where the predictions can be reviewed, and the model retrained with new data to enhance its accuracy over time.

There are many different machine learning algorithms that may be applicable to the data. Common algorithms for time series forecasting include:

* Linear Regression
* Decision Trees
* Random Forests

Each of these work in a somewhat similar way so no major difference to the development process if any one of them is chosen. Once the project is started a test will be done using each of the three to determine which is most appropriate for QuickStats output data.

When training a model, The project will split the rows of data into training and testing sets using a 70/30 ratio.

The selected machine learning algorithm will be trained on the training dataset using various functions.

After training, the model will be evaluated using the testing dataset. Metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared values will be computed to assess the model’s accuracy. Some of this data is what will stored in the Forcasting\_Log table in the Forcasting database.

If for some reason the accuracy or confidence of the predictions found from testing is low, an appropriate warning will be included on any resulting report using that model.

The trained and tested model will be used to make predictions for future dates. The forecasted predictions will then be outputted in the same structure and format as regular output data but into the forecast database.

It will also look for strong trends within the source data. Any notable trends will be recorded in the Trends table in the Forecast database.

A backlog of previous versions of the AI model will be kept and allow for reverting to previous versions if errors or inaccuracies occur.

**Example Analysis**

This is some examples of what the analysis process could look like. This is based off some real customer data:

**Observation:** Concetti and Powders show downtime percentages of 99.9% and 96.6%, respectively.

**Analysis:** Given this extreme downtime, regular active production is rarely achieved, suggesting the need for improved reliability.

**Recommendation:** Shifting from reactive to preventive maintenance on these assets could help address minor issues before they become major breakdowns, ultimately reducing downtime.

**Observation**

Many entries labeled as “DTM Info Not Entered” or “No Reason Entered.”

**Analysis**

The frequent lack of entered downtime reasons may stem from inconsistent data entry or limited training on the importance of tracking downtime causes accurately.

Comparing this to the rest of the data shows no specific operator or asset is to blame and it is a site wide issue.

**Recommendation**

Conduct training sessions for operators on the importance of accurate information and downtime logging.

**Observation**

Performance percentage values are sometimes over 100%

**Analysis**

This seems to be a regular occurrence with values commonly closer to 120%. This indicates that the targets are not accurate metrics.

**Recommendation**

A review of target outputs for running assets should be undertaken to improve resulting metrics.

# Databases

There are two databases types as part of ReportPro. The structured output which is a cleaned and formatted version of all the QuickStats output data, and a forecast database for the AI predictions.

While both of these databases are initially intended to only be involved with ReportPro it is possible other applications may want access to this data in the future so they will be made with this in mind.

The two databases will be similar in structure due to both mainly containing data in the QuickStats output format.

Each database will only have a single program that writes data into it but multiple that read from it.

## Output Database

The sctructured output database version of QuickStats output. The output database will be a very similar schema to the current QuickStats output database with a few key differences.

Data will be cleaned before it is inserted into the database. The data cleaning will involve the following processes:

**Handling Missing Values**: Filling in, replacing, or discarding records with missing values based on defined rules.

**Ensuring Consistency**: Ensuring the data aligns with the predefined structure and relationships in the database, including data type validation and field length checks.

**Removing Irrelevant Data**: Excluding fields that are unnecessary for the reporting documentation. This will greatly simplify the data and reduce the storage size.

**Fixing Wrong Fields**: Updating or removing data that is blatantly wrong (e.g., negative counts, decimal counts for int values, over 100% etc.)

The output database will be a single database for every year similar to QuickStats that is updated whenever new QuickStats data is received by the core data application. It will not create any duplicate records for already included data. The reason for seperating out data by year is to prevent an overloading of data in a single databases.

It is important to note that the only time existing records will be edited will be for the Down\_Time\_Reasons table if it has been updated.

Some headers may be renamed slightly from their QuickStats counterparts for better clarity or SQL compatibility.

## Forecast Database

The forecast database will contain future predictions for data based on the machine learning AI model.

The database will follow a very similar schema to the Output Database but all of the data will be a prediction.

The main difference between them is that every record in the forecast database will have a **log\_id**. This log ID will link the record to the forecasting log table and give information about how and when that prediction was made.

The log\_id is a one-to-many relationship. One log ID represents all the predictions made by one run of the AI model.

The Forcasting\_Log will record information such as:

* log\_id
* overall\_accuracy
* time\_taken
* date\_completed
* model\_version

The forecasting will not add duplicate records with matching key values even if the predicted results are different. This is to keep a standardised log of all the old, forecasted data.

Additionally there will be one more additional table within the forecast database, the Trends which will contain the following variables:

* trend\_id
* trend\_description
* trend\_strength
* start\_date
* end\_date
* log\_id

The Trends table will store identified trends and patterns derived from the model’s analysis of the dataset. It will include only those trends that are meaningful from a reporting perspective (e.g., excluding trivial relationships such as "faster speed leads to more output").

# Report Configuration

A report configuration program will be made for users to define what the resulting PDFs are like. Specifically it will include the following:

* Output location details (email, file directory, etc.)
* Desired pages/metrics in report
* Include forecasting checkbox
* Frequency and timing of reports
* Report colour palette
* Company logo

This will be a fairly simplistic Windows Forms project made using C++/CLI.

The user will be able to have multiple configurations for multiple different output report styles and structures.

Once updated the user will be able to click an “Apply” button. This will update the .expr configuration file associated with ReportPro within the reference folder of Prodigy and update.

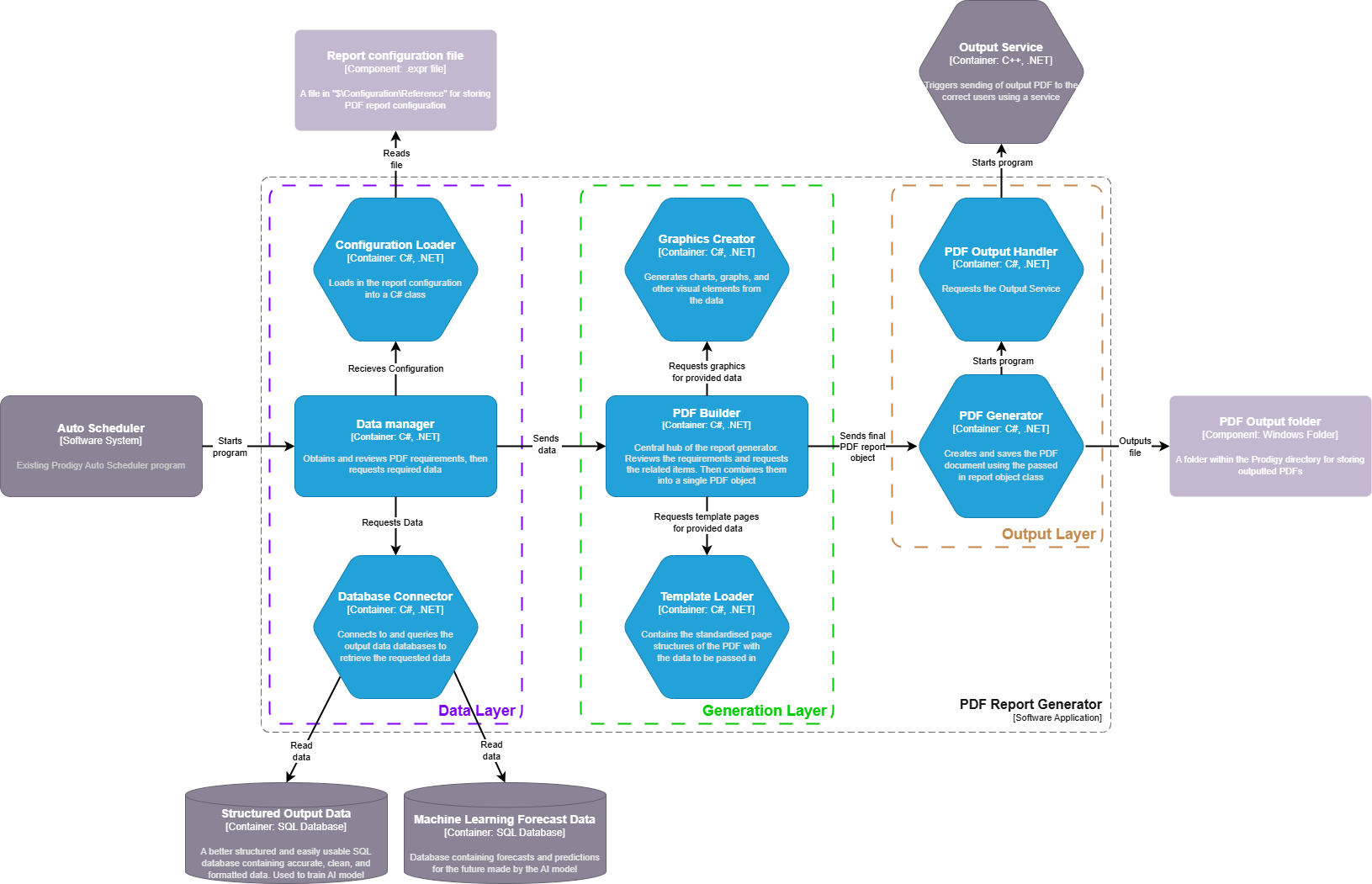
Some options for types of pages/sections to be included within the report include:

* Executive Summary
* Downtime Overview
* Root Cause Analysis
* Performance Against Targets
* Comparisons to Previous Periods

It will also update the Auto Scheduler with new information about when to run the data update and report generator.

# Report Generator

The Report Generator is most likely to be the largest part of the ReportPro in terms of development time. A level 3 C4 diagram of the PDF Report Generator is shown below:



The PDF Report Generation is most likely to be the largest part of the ReportPro. Because of this it was broken down further using the layer 3 model.

Each of the microservices shown are small parts of the project, mostly singular classes. Here are more details on each of them:

The project is split into a data, generation and output layer. These layers are to show the separation of the key stages of the report generation.

It is intended for each individual model to have unit tests and be able to each be tested as an individual component. This will make debugging and testing the program much easier and efficient.

The Report Generator program will make use of two third party libraries:

* LiveCharts – Used in the Graphics Creator – A .NET charting library for creating dynamic data visualisations like line, bar, and pie charts in C# applications.
* QuestPDF – Used in the Template Loader, PDF Builder, and PDF Generator – A .NET library for generating PDF documents, allowing for complex layouts, styling, and pagination in C#.

## Data Layer

**Configuration Loader**

Loads the data from the configuration .expr file into a C# class. This will be a single instance of this class that is accessed repeatedly so that the file is not opened more than once.

**Data Manager**

The Data Manager is the entry point for the Report Gen

The data manager will start by calling the Configuration Loader. It will then look at the resulting.

It next needs to get the data from connection configuration files within Prodigy.

It then analyses the loaded report configuration and makes the related calls to the Database Connector to get all of the data required for this report configuration.

**Database Connector**

A database connection manager that handles all connections and queries to any of the SQL databases.

This will be done using standard C# SQL libraries.

## Generation Layer

**Graphics Creator**

This uses the LiveCharts to generate the charts as pngs. The pngs will then be accessed by the PDF Builder to include them in the final PDF report.

The chart PNGs will be saved in a folder inside the Prodigy/Data folder. A sub folder specific to this report will be used.

There will be a separate, generic methods for each style of graphic, something similar to the following:

String^ CreateBarChart(String^ title, String^ xAxisTitle, String^ yAxisTitle, Data^ data) {...}

String^ CreatePieChart(String^ title, List<String^>^ Legend, Data^ data) {...}

**PDF Builder**

The PDF Builder is the central part of the report generator. It will create the list of pages which will end up becoming the PDF itself.

It will make requests to both the Graphics Creator and Template Loader to aid in building the list of pages.

The PDF builder will also handle the data that has been provided to it from the Data Manager. It will look at what pages are needed to be made and then begin from the first page creating each one. It will send appropriate data to the Graphics Creator and Template Loader when necessary.

Once completed the PDF will bundle the pages into a neat document object and send it to the PDF Generator to be outputted.

**Template Loader**

The template loader will be where the basic setup pages of reports are kept. It will return the completed page to the PDF builder upon request. This will make heavy use of the QuestPDF library to create the pages. Here is an example of how that could work for one page:

Page^ CreateOeeOverviewPage(Data^ OeeData, String^ OeeChartFilePath)

{

Page^ page = new Page();

page.Content =

{

Text("Availability: " + OeeData.AverageAvailability.ToString() + "%"),

Text("Performance: " + OeeData.AveragePerformance.ToString() + "%"),

Text("Quality: " + OeeData.AverageQuality.ToString() + "%"),

Image(OeeChartFilePath)

.Width(500)

.Height(300)

};

return page;

}

## Output Layer

**PDF Generator**

Takes the document object which has been passed into it and outputs the final PDF file to the desired directory.

**PDF Output Handler**

The gateway between the final PDF Generator being outputted to the call for the output service.

The handler will take the output PDF file path, and desired output configuration data and pass both of them to the service.

# Output Microservice

The output microservice, as the name suggests, is a small and simple program to output the resulting PDFs to the correct locations.

The service will be started by the PDF report generator itself. Once the report is outputted it will start the service and give it the location of the file and the information relating to output from the report configuration file.

The output service itself simply just checks what has been configured, creates the correct commands for the desired outcome, and starts other prodigy services using the correct arguments, namely the email manager, to send the PDF to the desired location.

The service will be made using C++ as it will then be able to easily integrate with existing Prodigy services.

The outputting service is kept separate from the PDF generator to establish a clear division of requirements, smoother integration with Prodigy services, and enable easier addition of new services in the future.

The service avoids direct access to the configuration expr file to minimise the number of locations requiring updates and to reduce the risk of the file being locked when a user attempts to edit it.

Additionally, this means that in terms of unit testing the PDF generator to be tested completely.