**Handover of HHH Development**

First, a huge thanks to the person(s) taking on the responsibility of HHH. It’s been in ‘shadow’ development for a while now and deserves attention from a real IT professional rather than a Google-as-I-go Product Owner. I’d also like to apologise in advance – for you will encounter various examples of bad non-sensical code. I am not a developer by nature, but I kept the code as clean and concise as my non-developer mind possibly could have, so please allow for areas of sloppiness. Whenever I revisit the code, I always find something to clean up, so I envision you will too.

**Preamble (Optional read)**

To give you background on the project, it started off as a way for me to skip out on a lot of manual work I was doing back in the days as an Account Manager in Debtor Finance. I used to spend hours on end doing manual allocations into Aquarius, filling out repetitive spreadsheets and other brain-dead tasks I wouldn’t consider real work (opinion only) - but had to be done one way or another. HHH started off as a collection of small scripts in a GUI, first in AutoHotkey (a macro maker if you will), then slowly progressed with VBA and even a bit of COM. *Hugh* success. I saved myself (oh and Moneytech) hours of hard, grueling manual labour every single day. It allowed us to scale up business immensely in the Debtor Finance department without hiring any extra operations staff for a significant amount of time. Thus HHH = Hughie’s Help Hub, a utility that lives to make the lives of Account Managers less of a living nightmare, at least on the processing side was brought to life (this sentence makes sense, I promise).

I wiped my hands and patted myself on the back and decided to pursue the Product. Low and behold – one of the first responsibilities I realized I needed to complete, before really running with the whole Product thing, was completing HHH and making it friendly to all users (that is – bug-free, stable and easy to use). So much of the Debtor Finance process depended on it. I couldn’t consciously leave it as it was (as in all honesty it was quite a mess, and only worked as a process at all because I was the one using it. Only I could fix bugs in it and only I could advance features. Not ideal for a growing business)

So rather than leaving the code as it was, and risk sending the whole reliant Debtor Finance process back to the stone age of manually chiseling each and every daily debtor receipt into the stone tablet that is Aquarius (which, in all honesty is what we did before, and what our major competitors do to date), I decided with huge help from Chen in the Accounting team, to migrate the full code to Python, whilst upgrading the process to be even more automatic and also integrated Chen’s Help Hub, or **mPower**. The vision was to get a seamless data flow between the Debtor Finance operations team and the Accounting team, who oversee liquidity and moving money to the correct places.

Given the nature of my role, constantly tending to the delicate baby that is Aquarius, completing ad-hoc higher-priority-than-HHH projects and what have you, this vision sat on the backburner for quite some time, and was only worked on by myself when I was freed of everything else. If my role was the economy, then HHH would’ve been the beggar right at the bottom just waiting for that waterfall to trickle on down. After chipping away at it for months, the stars have aligned, and HHH is now being moved to the IT department.

Oh, and did I mention we slapped on our Equipment Finance account management into HHH as well? Yeah, that and emailing our mass mail merges.

**Business Decisions**

Given that the project started off non-official, but now performs a lot of business-critical functions, there exists a decision to be made about the future of HHH. Do we:

* Stop the development and maintenance of HHH, and find other solutions for the business functions?
* Continue developing HHH in Python? (there are a lot of places this piece of software can be taken)
* Migrate the code to another language?

The formal decision is still to be made by the relevant team leaders.

**Stakeholders**

Here I’ll name (and thank) the primary stakeholders of HHH:

Smruthi Shreepathi – Primary SQL developer

Zhipeng Chen – Owner of mPower, which HHH is integrating into

Hugh Huang – Owner of HHH

Mark Cameron –Primary Report – Business

Bernard Sircelj – Primary Report – Software

**Deprecated Functions**

Old HHH used to have the following, but I’ve deprecated them, due to priority and because they are not too impactful:

* Shortcut scripts to important screens in AQ, ANZ and WPC, and client folders in common
* Shortcuts for specific repetitive tasks such as checking AQ for out-payments, checking AQ for new debtors and schedules, or searching up a particular client
* The ability to bulk import verification statuses of invoices and have them updated into AQ

The Debtor Finance End of Day process used to also sit in HHH. After Aquarius had finished updating its figures of all the days payments and account movements, a query was made into the AQ database to pull relevant movement figures out into the AQ Movements Master File (that also contained the debtor receipts from the morning). This file was then being automatically sent to the Accounting team so they could process accordingly.

With Chen and Smruthi, we migrated this process to sit in the mPower engine.

**Current Functions**

Here I’ll break down each function of HHH and because it hasn’t been formally deployed yet, what the known issues are, and what remains to be developed.

Segment Reports

Certain Debtor Finance (DF) clients need their segment reports sent to them daily, as they do not have access to the WPC banking portal.

*Old HHH*

AHK script scrapes WPC website to download reports to directory ([U:\Customer Filing\Debtor Finance\a Segment Accounts (All)](file:///U:\Customer%20Filing\Debtor%20Finance\a%20Segment%20Accounts%20(All))), then email is generated through COM and displayed on Account Manager’s screen for review/sending.

* Known Issues
  + Some timeout issues occur when the WPC website is slow
  + Code crashes when a password reset is required

*New HHH*

Basically, just a migration of the AHK code to Python, using the Selenium module.

* Known Issues
  + Same as above – started looking into the password reset, have not resolved yet. WPC only accepts a limit character count, which ruined my solution to append the date to the end of the user’s password when it needed to be updated. Note, opportunity to test and fix is small, as password reset happens only once a month.
* Further Development
  + Allow users to select the date they wish to extract report for. Currently only handles ‘today’.

*Status*

The current process still lives with *Old HHH*.

Debtor receipts

A daily morning process that takes the mPower calculated figures for our customers and imports them to Aquarius (AQ, Debtor Finance platform).

*Old HHH*

mPower calculations are output into a spreadsheet, and an AHK script is run to loop the spreadsheet rows and macro them into the relevant AQ screens one by one.

* Known Issues
  + Certain lag-induced errors can stop the script from running or even mess up the figures entered into the system. Although rare, it is a pertinent risk.
  + Code involves several steps (used to be necessary, but not any more since the deprecation of the AQ Movements Master File spreadsheet) of importing data to different locations, which has in instances (again, rare) caused data discrepancies.
  + If the loop is broken, no automatic way of picking up where the code left off.
  + Inherently slow due to the need to wait for AQ to load screens and process data. Current process takes circa 5-10min

*New HHH*

mPower calculations are pulled straight into a Pandas dataframe for direct parsing into AQ. AHK is still called upon to do the macro inputs, however, the code has been improved upon to be much more reliable, and the underlying process in AQ has been changed, so that no incorrect data can be input, and if the loop breaks, the code can be run again where it left off.

There is also the function to reverse all the imported receipts for a day, if ever required, and multiple users can run the code at the same time to reduce the overall run time down to under a minute.

* Known Issues
  + Certain select and extremely rare situations can break the AHK loop, thus requiring the user to start it up again. But not particularly risky, as it won’t cause incorrect figures to be allocated to clients. This is hard to solve for, considering AQ can only be automated internally by scraping the keyboard and mouse.
* Further Development
  + Needs to be tested in live environment.
  + Continue to consider new ways to achieve the debtor receipts import without AHK.

*Status*

The current process still lives with *Old HHH*. Testing needs to be complete for the live environment before it is migrated across.

Back-to-Back (B2B)

We have a couple of B2B clients that exist outside of AQ that need to be transacted with and managed daily.

*Old HHH*

This never existed in HHH. It was manual spreadsheets and emailing of transactions to the accounts team.

*New HHH*

Transaction details are input into a screen before being committed to the mPower database for processing.

* Known Issues
  + Currently, it is cheekily just sending an email in the background to accounts with the transaction details (a short and sweet way to start removing the manual spreadsheet involved).
* Further Development
  + Remove emailing, and direct transaction inputs to B2B designated database
  + Add in facility management tools like movements, fee structures, certain conditions (pending formal decision)

*Status*

Still manual. Decision required on the future of B2B management.

Mail Merger

No suitable tool existed in the market to do mail mergers on our terms, so I made one. Mainly for Notices of Assignments, it generates a customized notice per debtor, and attaches it to a personalized email to be sent out automatically.

*Old HHH*

The NOA was a fixed attachment, alongside an email that was sent out to recipients in an Excel mailing list.

*New HHH*

Basically the same, but added the feature to personalize both the email and the attachment, as well as save to PDF. It was also faster thanks to Python multi-threading (about 1-2 seconds per merge and mail).

Simply upload document with variables enclosed by “<<var>>”

Upload Excel data, with each column as a different variable

Match variables with document variables (if they are not the same)

Select merge options – If and where to save, what format to save, what the recipient variable is (along with cc and bcc) and the email body (which accepts variables too).

Do a test merge to the user’s own email, then complete the merge in a background thread.

* Known Issues
  + Email body font does not always turn out exactly as previewed (html issues).
* Further Development
  + Add in option to print merged documents (in case we have physical mail merge in future).

*Status*

The old one is no longer being used, and we are in-flight to deploying the new one. It’s been used a few times already for merges of different natures.

Equipment Finance

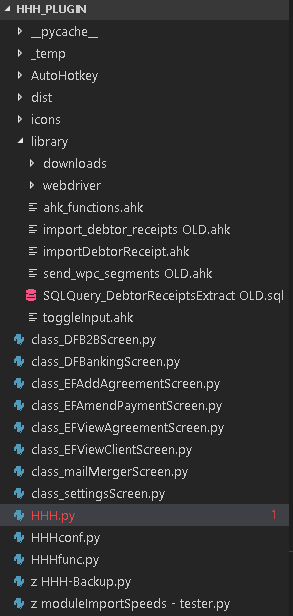
We recently introduced a new product to our line of financial solutions – Equipment Finance. As a result, I worked on a facility management tool for it, and built it in HHH.

* Current features
* Add Agreement – generate payout schedule and add client details
* View Agreement – all live agreement details as well as payout schedule
* Amend Payment – add and amend subordinate charges associated with the scheduled payment.
* Start and End of Day (incomplete) – for receiving and communicating movements to mPower. Should be automatically scheduled with mPower
* Known Issues
  + Timing Issues involved with how subordinate statuses are updated from due to overdue, and what subordinates to include in the end of day processes (prevent doubling up on subordinate payments)
* Further Development
  + Maintain Agreement (incomplete): for editing agreement figures, generating statements, calculation of payouts, terminating or re-rolling facilities etc.
  + Calculate initial interest charges between settlement and agreement start date.

*Status*

Not live but being used in parallel with a few live accounts for testing.

**The Code**

****

Class structure breakdown

HHH.py

* MainApp – When code is initialize, it calls this module to load the mainApp window and all it’s widgets. It is then connected to other ‘class\_’ modules that loads up the relevant screen when the user clicks the button

HHHconf.py

* widgetEffects – animations and effects of widgets go here, such as flashing or fading messages, or making windows transparent
* HHHWindowWidget – The base class for all ‘class\_’ screens and their sub classed screens. Any new screen should call this base class, and it serves like a template to add widgets to.
* dragDropButton – this is currently used in the class\_mailMergerScreen for users to drag and drop files onto a button
* tableComboBox – custom comboBox used specifically in tables
* CustomMenu – the context menu used across the whole app, that comes up when right clicked on certain widgets (tables)
* HHHTableWidget – custom template table used for almost all tables (not in mailMergerScreen at the moment, but work should be done to migrate these tables)
  + pandasModel – populates Pandas dataframes into table models for display
* calendarLineEditWidget – lineedit class that pops up a calendar widget when focused
  + customCalWidget – purpose built for the calendarLineEditWidget
* HHHTextEditWidget – allows user to tab past the widget when tabbing. Normal widget sends the tab into the text
* HHHComboBox – dropdown widget that can be controlled nicely using the keyboard.

HHHfunc.py

* mainEngine – methods that control calculations or data for the app in general
* mailMergeEngine – methods related to the mail merger
* debtorFinanceEngine – methods for debtor finance
* equipmentFinanceEngine – methods for equipment finance

class\_B2BScreen.py

* b2bScreen – called by button on main app

class\_DFBankingSreen.py

* bankingScreen – called by button on main app
* segmentsWorker – used as a threader to run in the background. Called by bankingScreen
* receiptsWorker – used as a threader to run in the background. Called by bankingScreen

class\_EFAddAgreementScreen.py

* EFAddAgreementScreen – called by button on main app

class\_EFAmendPaymentScreen.py

* EFAmendPaymentScreen – called by clicking table row on main app and in EFViewAgreementScreen
* subordinateEditScreen – called by clicking table row on EFAmendPaymentScreen
* subordinateAddScreen - called by button on EFAmendPaymentScreen

class\_EFViewAgreementScreen.py

* EFViewAgreementScreen – called by clicking table row on main app
* maintainAgreementScreen – called by button on EFViewAgreement Screen

class\_EFViewClientScreen.py

* EFViewClientScreen – not called yet (incomplete)

class\_mailMergerScreen.py

* mailMergerScreen – called by button on main app

class\_settingsScreen.py

* settingsScreen – called by button on main app
* passwordUpdateScreen – called by button on settingsScreen
* appCredentialsScreen – called by clicking table (or context menu) on settingsScreen

Module Descriptions

*\_pycache\_*

Cache, I guess.

*\_temp*

Directory for temporary files generated in the process of HHH modules, such as the mail merger. They should be cleared out when HHH closes.

*AutoHotkey*

The AHK exe that HHH uses to run AHK scripts with. The scripts are left uncompiled so that developing with them is easier.

*dist*

Where HHH exe is ultimately compiled to. Eventually, exe shouldn’t exist as it will be integrated into mPower as a module.

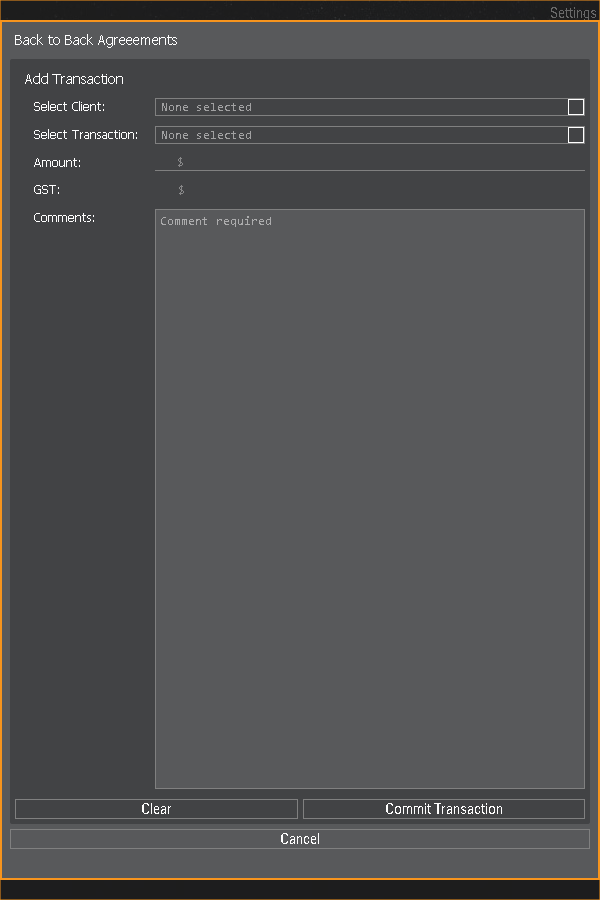
*Icons*

Images used by HHH

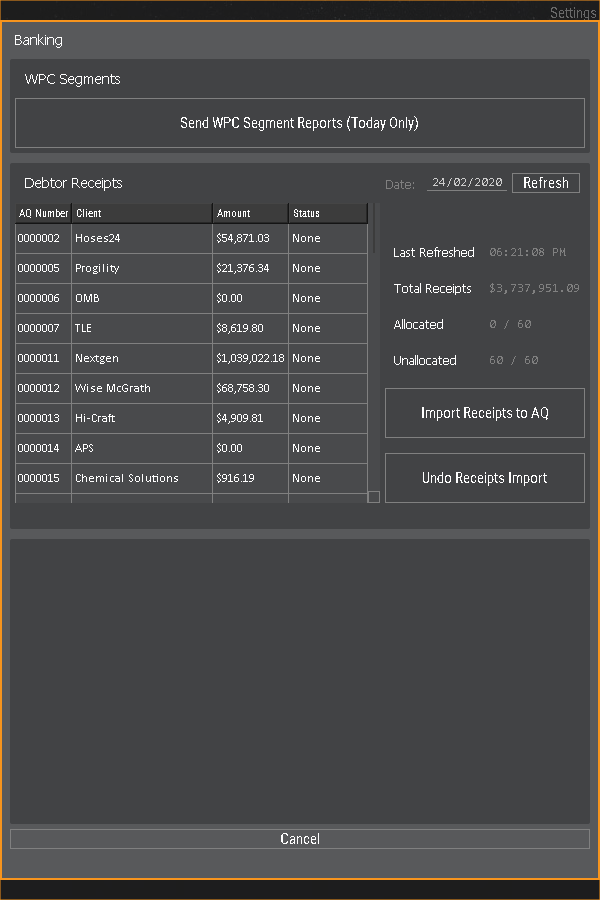
*Library*

Resources used by HHH

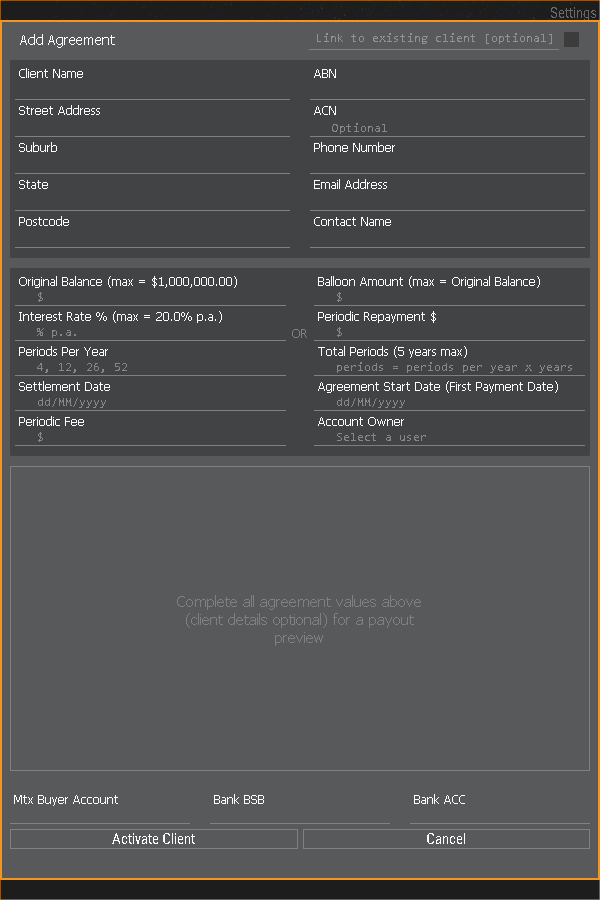
*class\_DFB2BScreen.py*



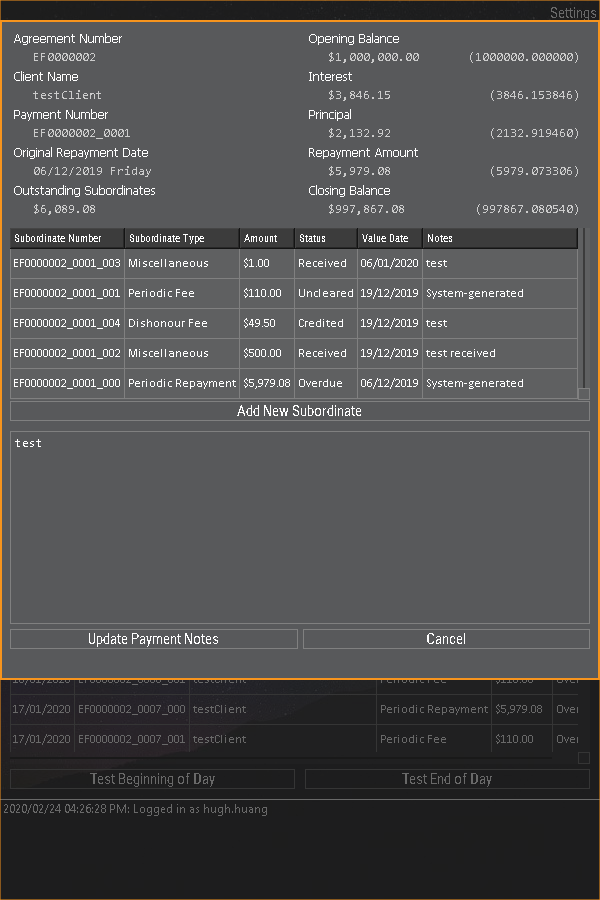
*class\_DFBankingScreen.py*

**

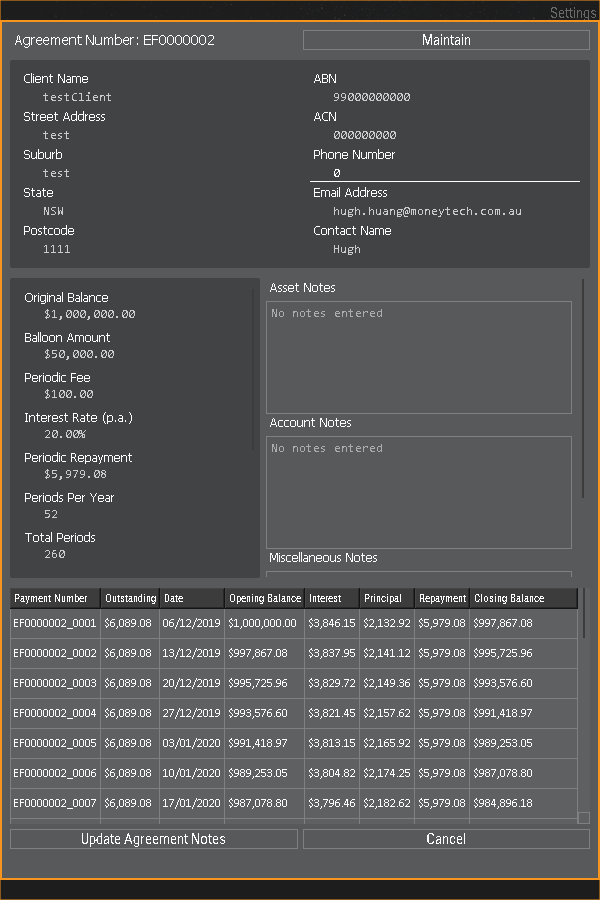
*class\_EFAddAgreementScreen.py*

**

*class\_EFAmendPaymentScreen.py*

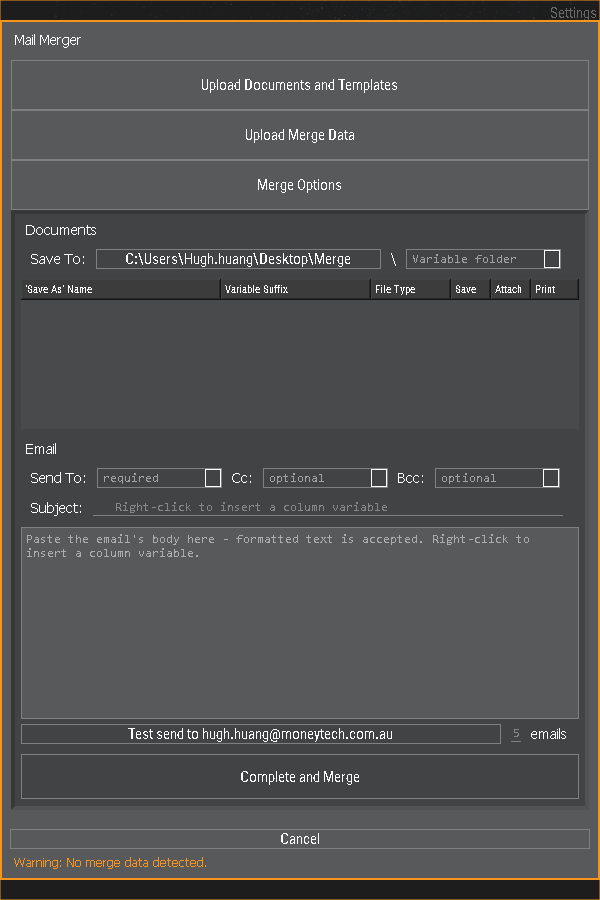
**

*class\_EFViewAgreementScreen.py*

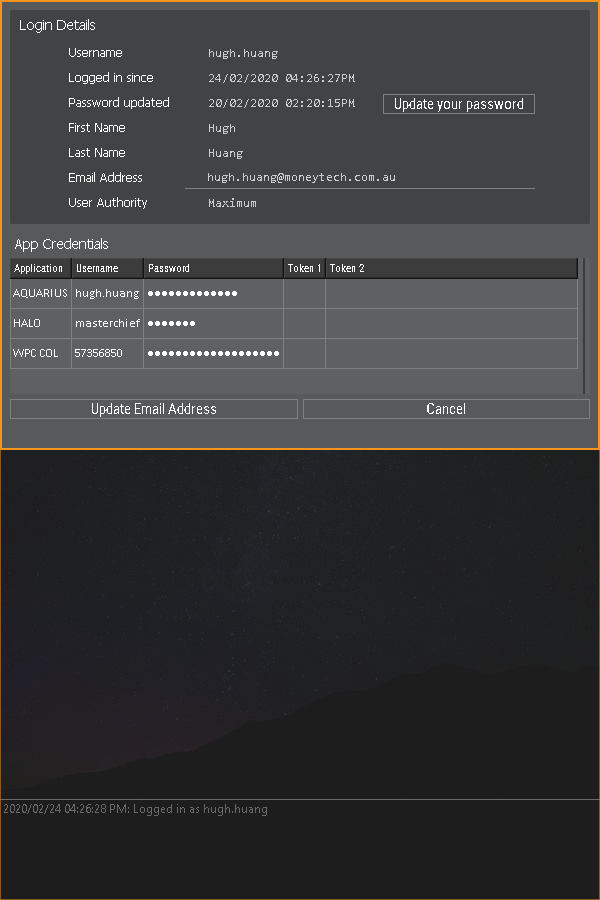
**

*class\_EFViewClientScreen.py – Incomplete*

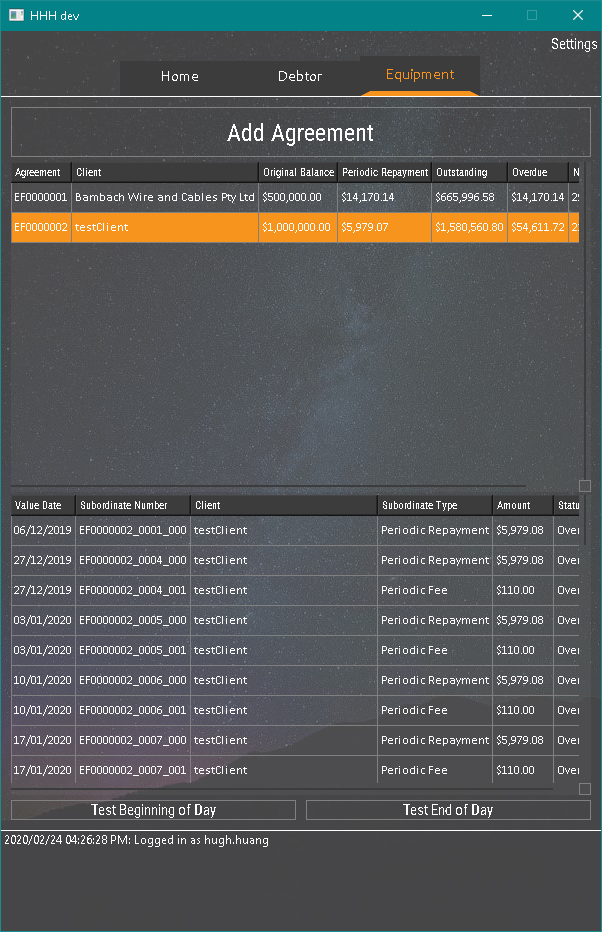
*class\_mailMergerScreen.py*

**

*class\_settingsScreen.py*

**

*HHH.py*

**

*HHHconf.py*

This module contains variables and configurations that in future, some should be made available to users to edit in the GUI. It also contains custom PyQt5 classes that all the previous screens use.

*HHHfunc.py*

This module contains all the main functions that manipulate or extrapolate data from the database. The screens are just screens, this module is really the engine. I do note, that there are a few remaining methods in some of the ‘screen’ modules that really belong in this engine module. You’ll probably notice them straight away as they do important calculations, or move data to databases etc.

*Z*

Old code and testing code

**What’s left to do**

Obviously, I have a dream. I’m not too emotionally invested in this dream to be dramatic if it doesn’t come true, but below is what I had listed down as the remaining tasks for the project. Everything is of course up for discussion from a business requirement and a prioritizing perspective. The intention of this was always to be a running utilities project, where certain operational tasks could be automated and collated into one place. Keep that in mind as it may stay the same or it may change. I’m vague like that.

|  |  |  |  |
| --- | --- | --- | --- |
| HHH | Deliverable | Task | Notes |
| EF | EOD | Push due transactions into db | Dataframe constructed and ready for db push – wait for Chen’s availability before commencing mPower integration of EOD. Also requires locking of uncleared payments |
| EF | BOD | Obtain dishonour info and use to update UNCLEARED status | Currently awaiting eod integration before receiving dishonour info from mPower to update statuses |
|  | Notifications Centre | Notification UI + email receipts to users | User to receive notification on important changes to their account or clients |
| EF | Attach files | To consider cloud for file management | File management for facilities |
| DF | B2B transactions | Create transaction screen to push manual B2B requests into AQ Instruction table | Also consider facility management, such as movements, risk, actions, file management, availability calculations |
|  | mPower integration | Non-negotiable | Must be completed. |
| EF | Client Statement | Generate statements |  |
| EF | Payout Figure | Generate payout figure based on date |  |
| EF | Termination | Ability to terminate facilities |  |
| EF | Settlement Interest | First transaction – interest between settlement and first payment date |  |
| DF | AQ shortcuts |  | Optional |
| DF | Account Manager metrics |  | Risk metrics and account behavioural analytics with flagging system |
|  | Audit trail for system | Create db table that tracks changes to things in HHH | Such as, adding or changing new agreements, running receipts, requesting transactions etc |

Other than that, I’m out – well, I’m not out, I’ll still be around as the project’s Product Owner, which actually means I’m in, very very in. We’ll be working closely together to push this project in the direction Moneytech wants it to go. Of course, after this read I’ll give you a demo of the app so far, break it down even further, and answer any asks you may have.

But other than that I’m out.

Hugh Huang