**Avcol Forms**

Technical Details and Write Up

**Avcol Forms App**

[GitHub Page](https://github.com/AvcolDJPownall/AvcolForms)

**Introduction**

My desktop application aims to provide an easier method to fill out school-related paperwork. Both students and teachers will be able to access various forms using the school computers. The implementation is backed by a Windows Form application acting as a graphical interface, along with integration with an SMTP client for delivering submissions to the relevant departments. These aspects aim to mitigate the school’s reliance on physical paperwork where digital alternatives are practical, greatly reducing costs associated with printing and overall environmental impact.

# **Analysis**

**Client**

Avondale College primarily uses paper forms for both students and teachers. This presents an inconvenience to many users, as finding and printing the appropriate paperwork can require a large amount of time and resources. The solution I am presenting gathers many common forms and displays them in one digital repository. Since Windows 10 desktop computer are exclusively utilized across the school, this creates an interesting opportunity for a Windows Forms application.

**Specifications**

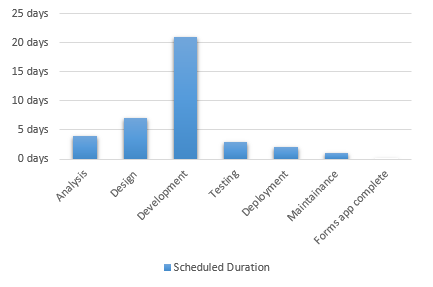
* Windows Forms application with GUI
* Utilizes .NET Framework 4.7.2
* Automatically submit emails containing the form’s results

**Relevant Implications**

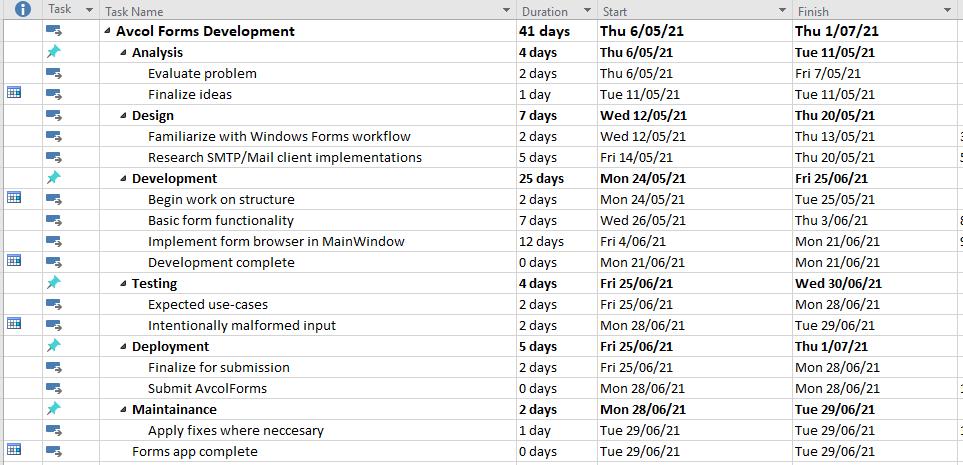
When designing software intended for widescale utilization such as across Avondale College, certain considerations must be made to limit potential issues. For instance, all art assets were designed from scratch to avoid copyright or licensing concerns. No trademarks or intellectual property licenses were violated in the design of Avcol Forms. I aimed to make the program as modular as possible to ensure a high level of future-proofing, which is especially important as this program would need to be integrated into the school’s mail server through a backend application. User privacy was also a large concern due to the public nature of the school’s computers. To prevent exposing potentially sensitive information, I decided that the program would not write to any database (local or otherwise) and would only share information with the email client. Form responses would remain strictly in memory until the form is ready to be sent to the network via an encrypted email (HTTPS). Aside from those carbon-copied into relevant email responses, no user data is shared with third-parties.

**Project Schedule**

Project management was handled through Microsoft Project, an industry-standard piece of software. Most of the project’s time budget was dedicated to development, as a proper implementation was needed to ensure that all intended features could be reliably implemented in the resulting software.



A week of research was needed to evaluate different methods of delivering form submissions to the relevant parties. As seen by this task sheet, a large amount of development time was dedicated to creating an environment for the user to browse forms. This required making a few art assets from scratch for UI elements such as buttons and other graphics. The basic form submission framework would take around a week of the development time budget, which includes the basic handling of attachments and designing a client for submitting SMTP requests.



# **Design (Graphics)**

**Flow Chart**

The application should act as a glorified SMTP mail client, as depicted by this diagram. The student and teacher user types both share the same form selection menu. Any checks or error handling performed when the user submits the form should provide the user with visual feedback, before returning to the base form.

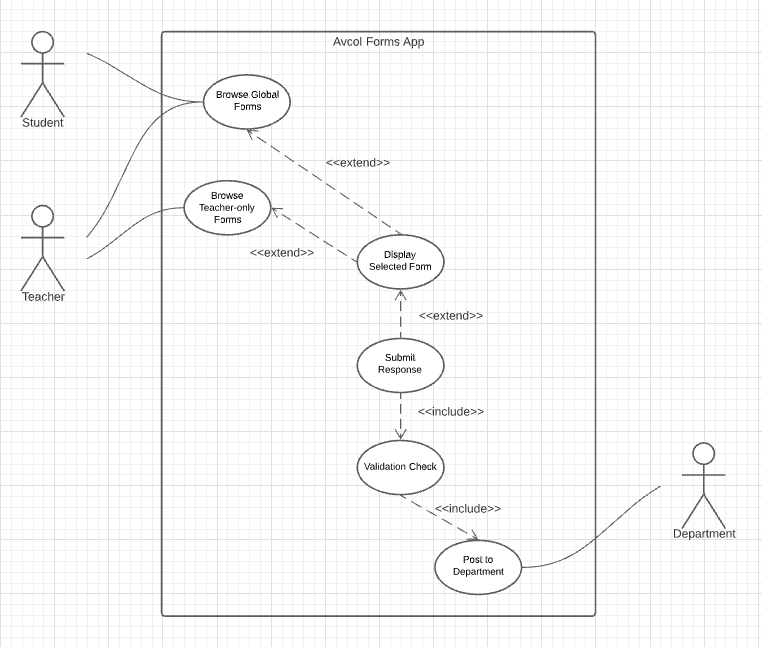
Diagram

Description automatically generated

**Use-case Diagram**

A basic demonstration of intended functionality.

Here we see two separate front-end entities, students and teachers. These two groups may access a repository of forms with contents tailored to their needs. Following a series of sanity and error checks, a submitted form should be posted to the specified department. Any STMP errors or incorrect form data will inform the user upon submission, returning them to the previously selected form. If no issues are found, all relevant departments are carbon-copied into an email via STMP.



# **Development**

Windows Forms revealed some design limitations and challenges in early stages of development.

For instance, many attributes cannot be modified without halting the main thread. This posed issues when working with events, as some elements needed to be modified when the form changed states. To remedy this, I extensively used the method Task.Run() to run basic operations on a new thread, thus allowing actions such as submitting forms to feature multi-threading.

Another issue I encountered was authorization inside the SMTP client. Storing the mail client’s password inside the program would result in security issues, especially considering how easy it is to decompile .NET Framework binaries using third-party tools. In an official implementation, emails should be sent from a server-side program, perhaps from a TCP connection or POST request. This would drastically improve security, as there would be no way to reverse-engineer the program to find the password string. While the program does not feature these measures, I’ve taken steps to ensure the password is not stored in plain-text. If the school’s IT department decide to use this program, they would need to implement a server-side solution rather than obfuscating the client.

Once basic SMTP functionality had been established, focus shifted toward visual design of the Avcol Forms application. Mock-up assets for certain UI elements were designed in Adobe Photoshop, establishing a good scale reference for buttons and banners. The main menu transitioned to Mdi documents, improving application flow and user experience. Scale was later adjusted to accommodate an A4/A5 paper aspect ratio, a design decision made to assist transitions from existing paper forms to digitally formatted pages.

# **Implementation**

**Testing**

During development, extensive testing was needed to eliminate bugs and unexpected behaviour for the final release. This testing sheet details a range of unexpected behaviour and challenges encountered in the preview stage. Most modifications could be linked to the user breaking intended functionality through unexpected inputs, though some user feedback indicated that behavioural bias needed to be considered to improve the user experience. This feedback was largely used to adjust certain UI elements and window layouts.



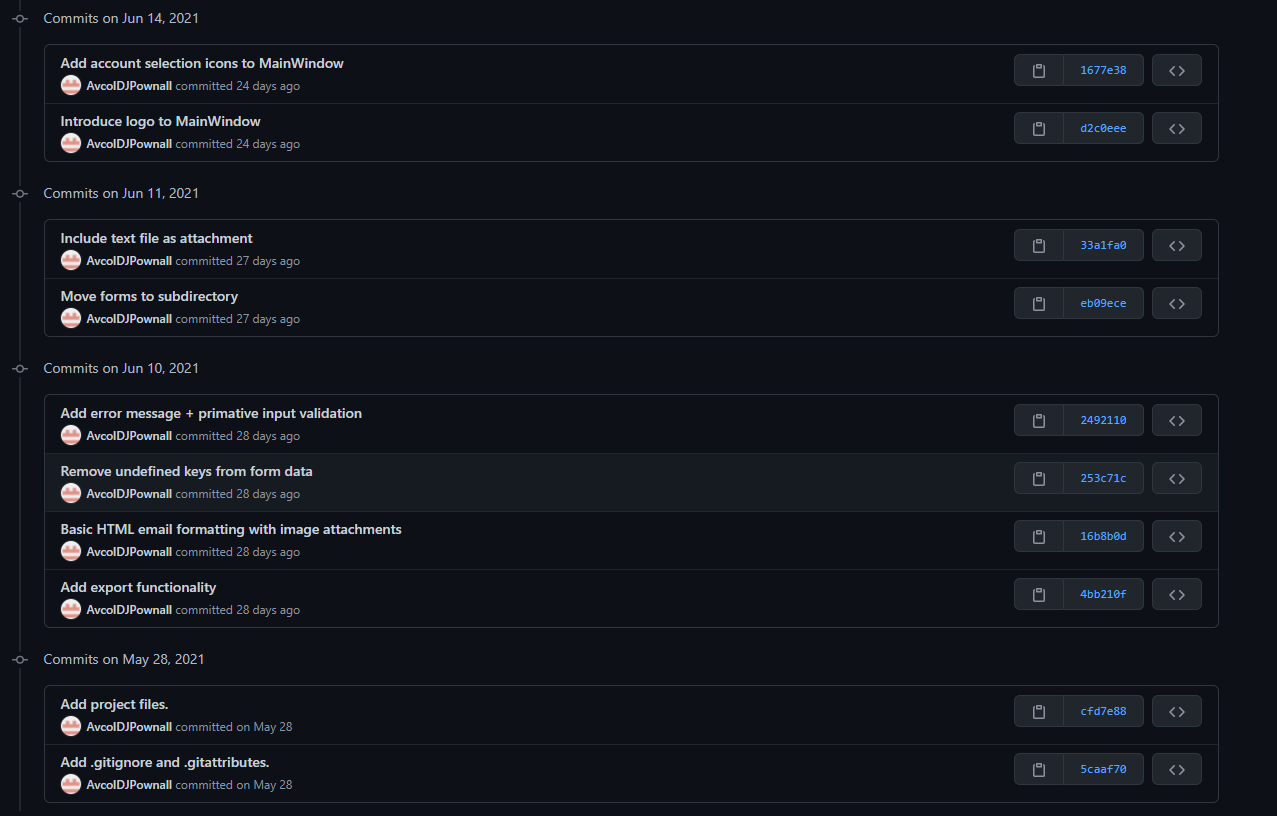
**Documentation**

A brief overview of the application along with basic contribution information can be found [here.](https://github.com/AvcolDJPownall/AvcolForms/blob/master/README.md)

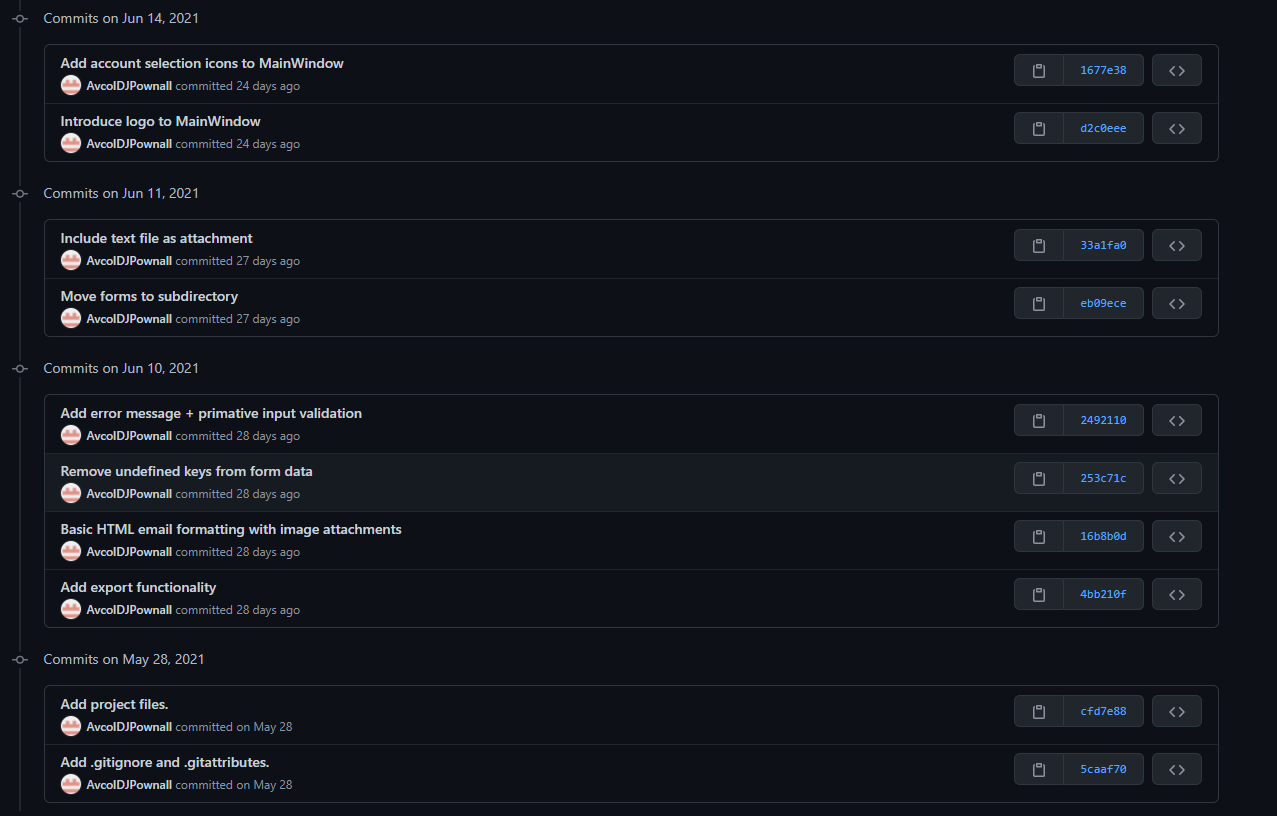
**Progress (GitHub commits)**

All version control and historic commits can be viewed on GitHub via [this link](https://github.com/AvcolDJPownall/AvcolForms/commits/master).

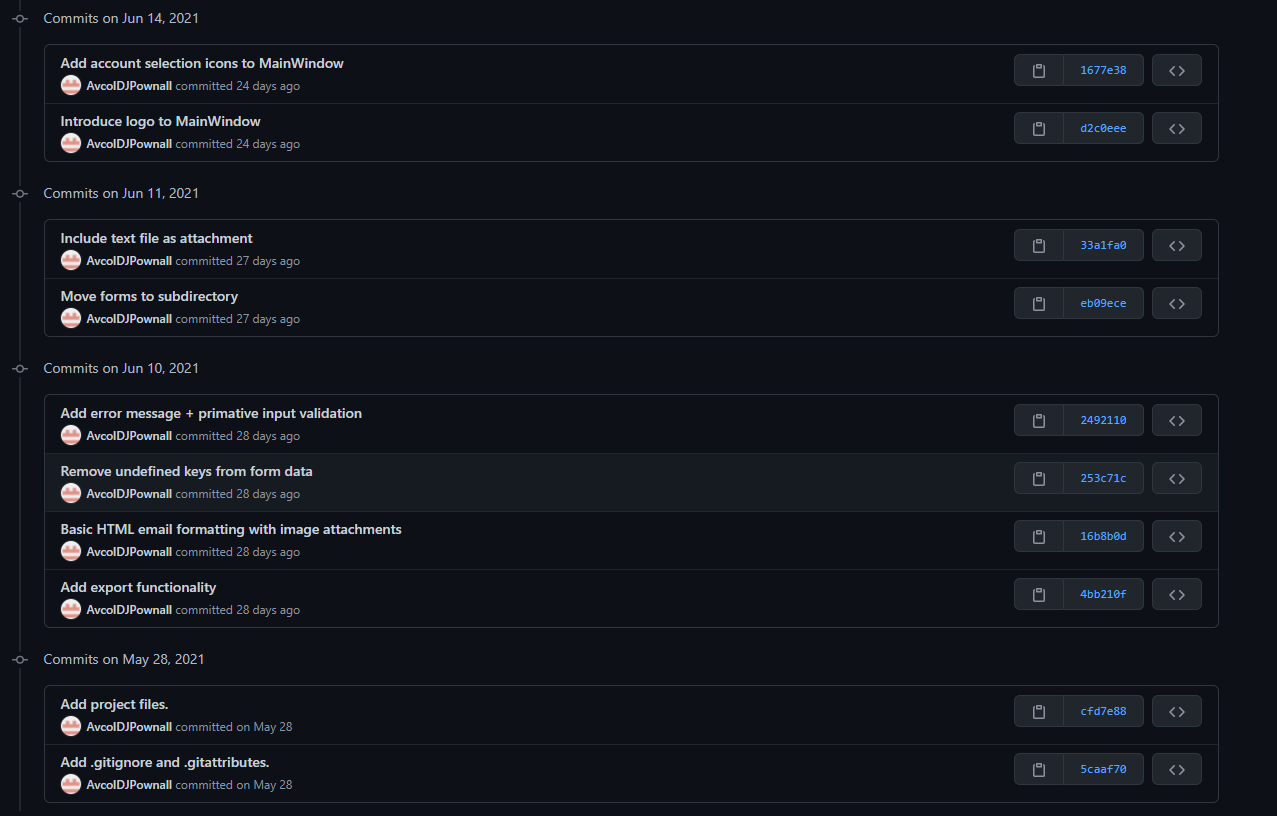
May 28 - Basic SMTP functionality was completed after some experimentation.



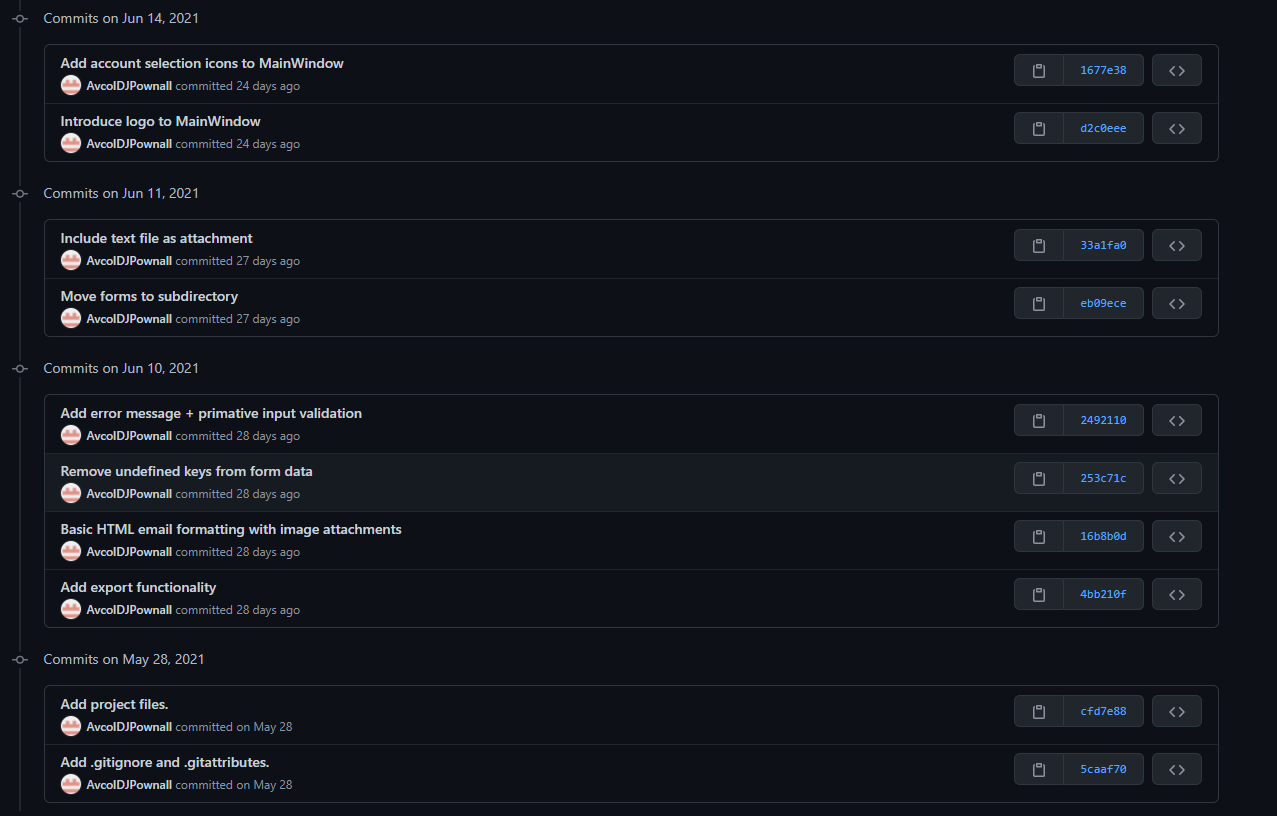
June 10 – Some experimentation with exporting Microsoft Excel data and back-end data handling



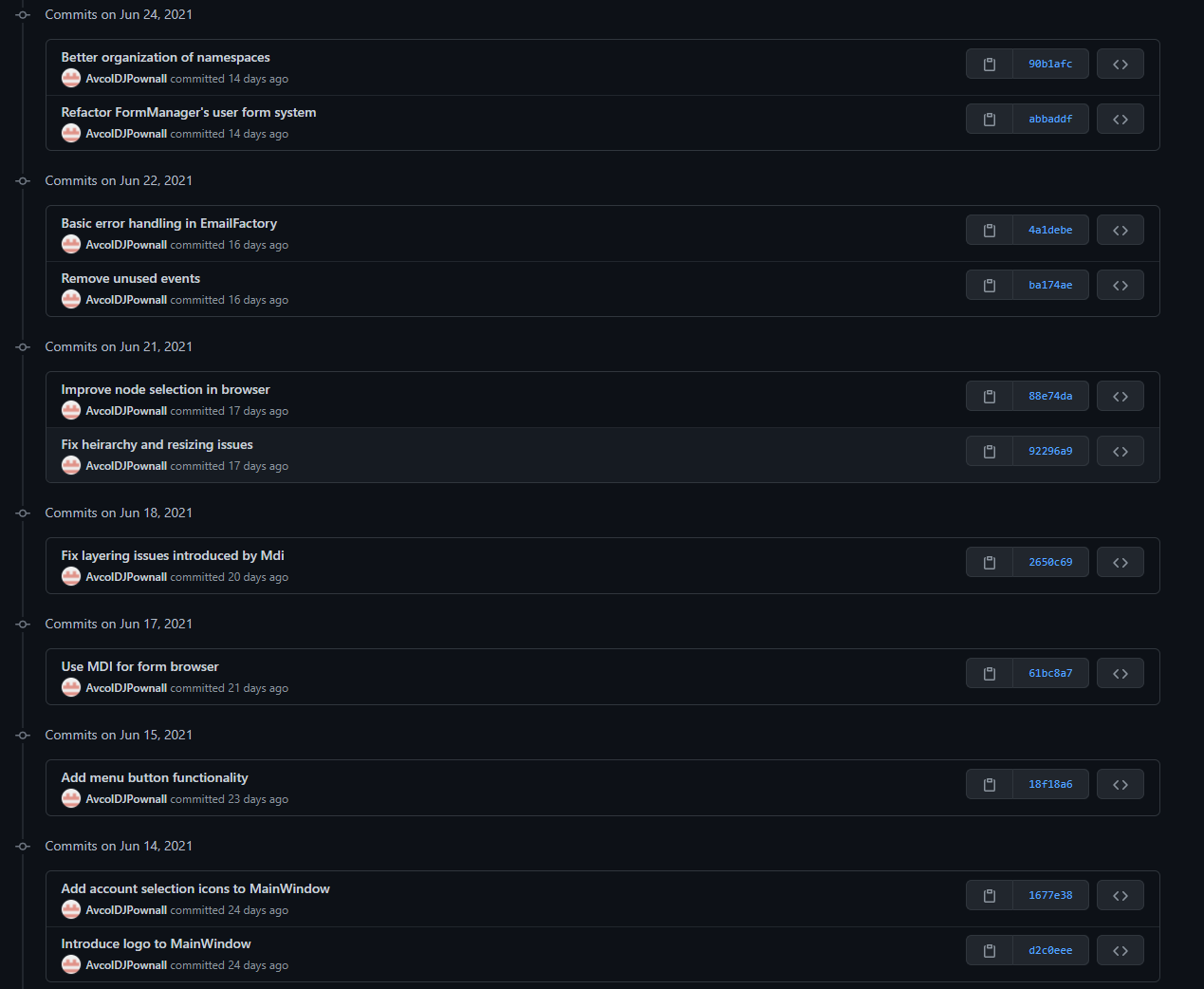
June 11 - Moved to a text file based data exportation method and implemented nested forms under sub-catergories.



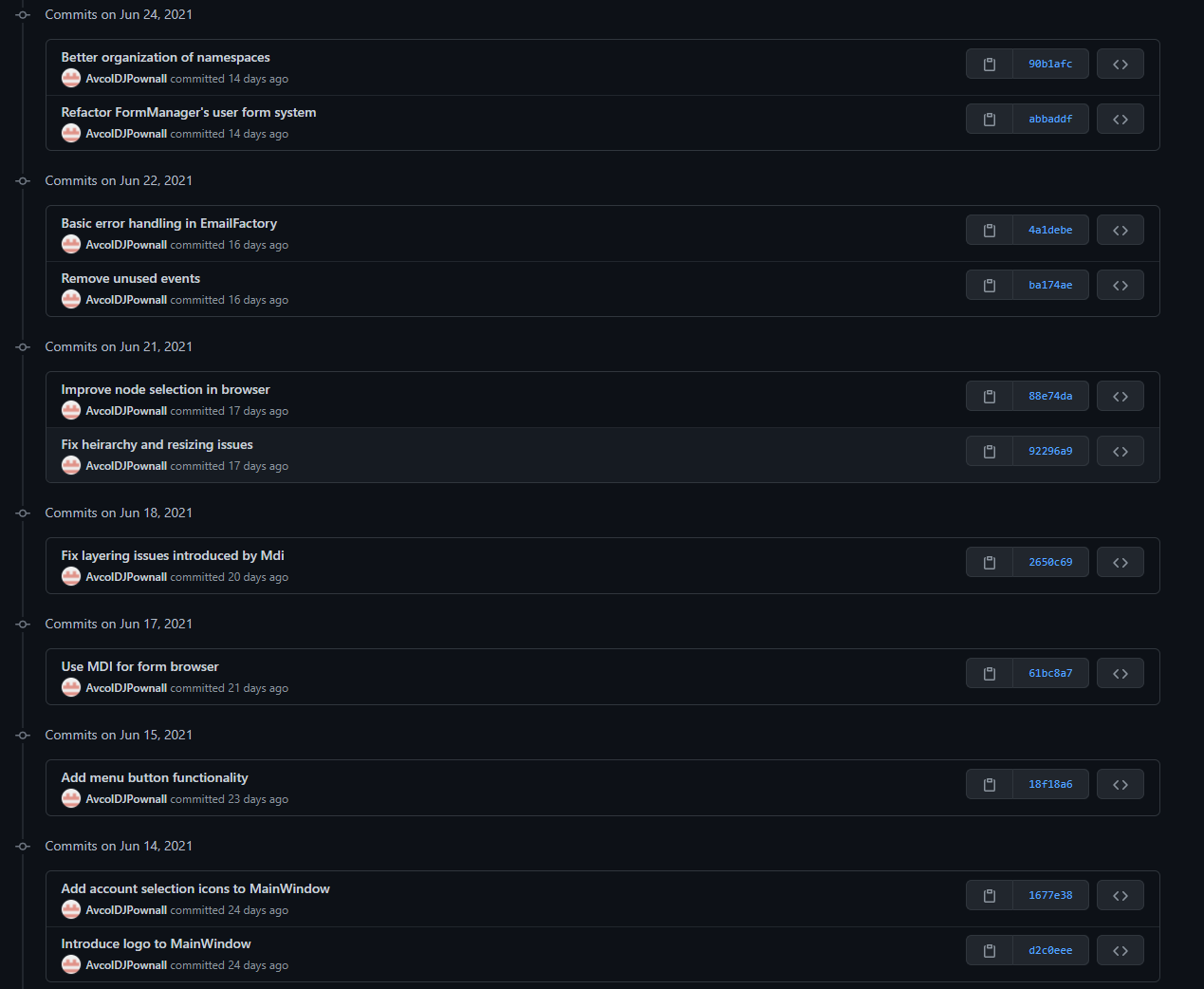
June 14 - This session was spent on the visual design of the logo and account type buttons.



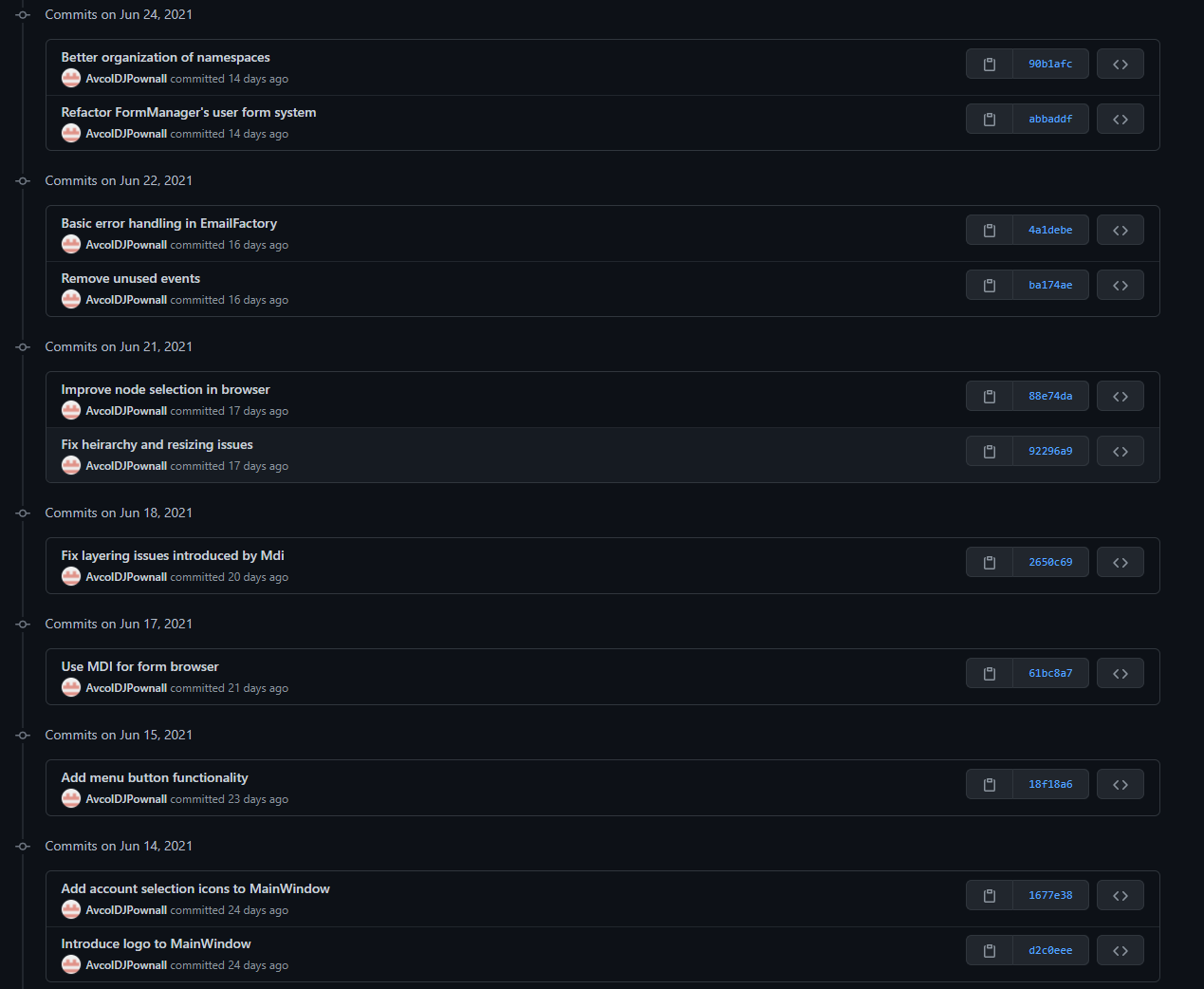
June 15 – Started work on account type selection, this would allow the user to choose between Teacher and Student forms.



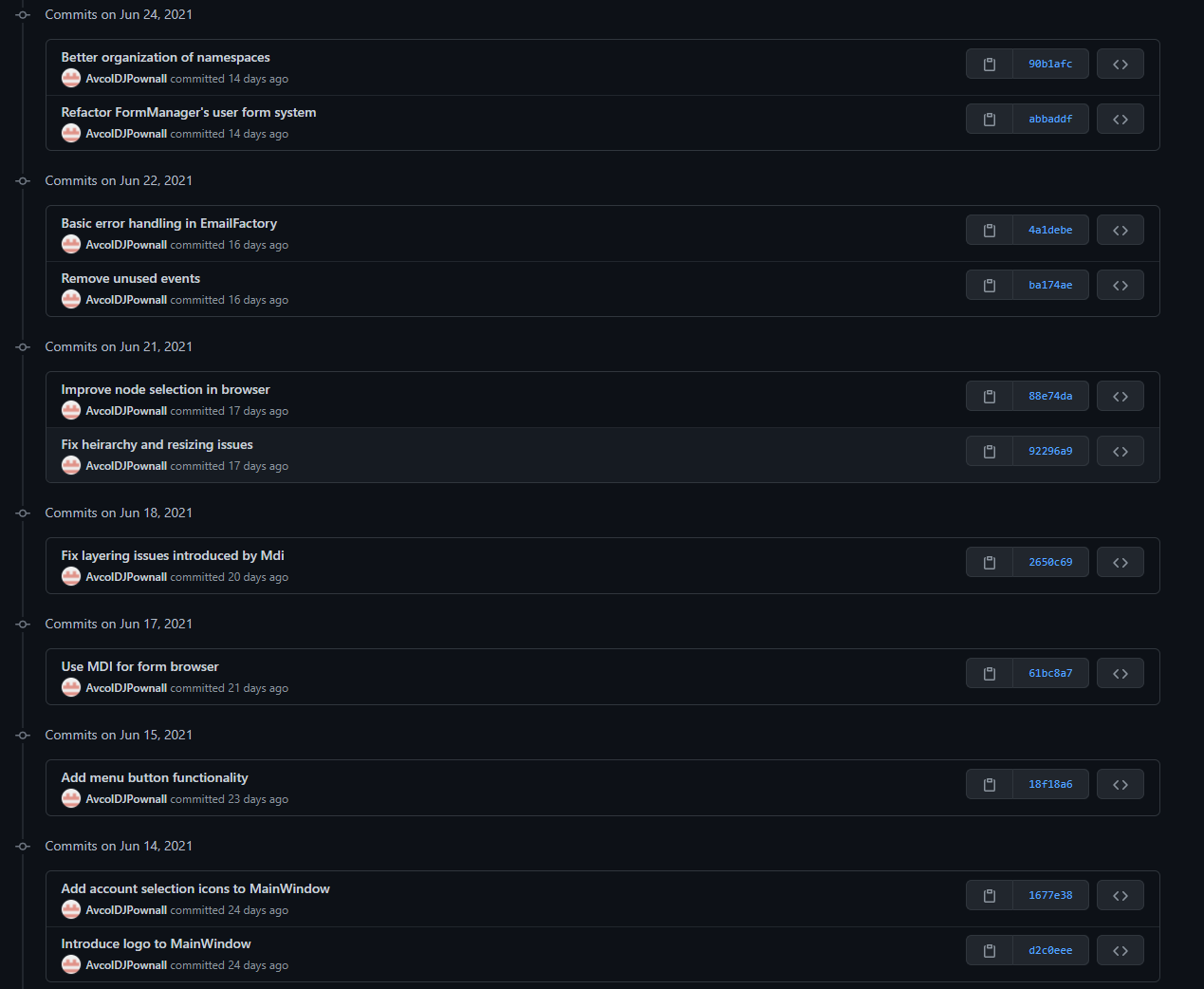
June 17 - Switched to using Mdi windows, as the form browser should use the same window as the main menu and account selection interface.



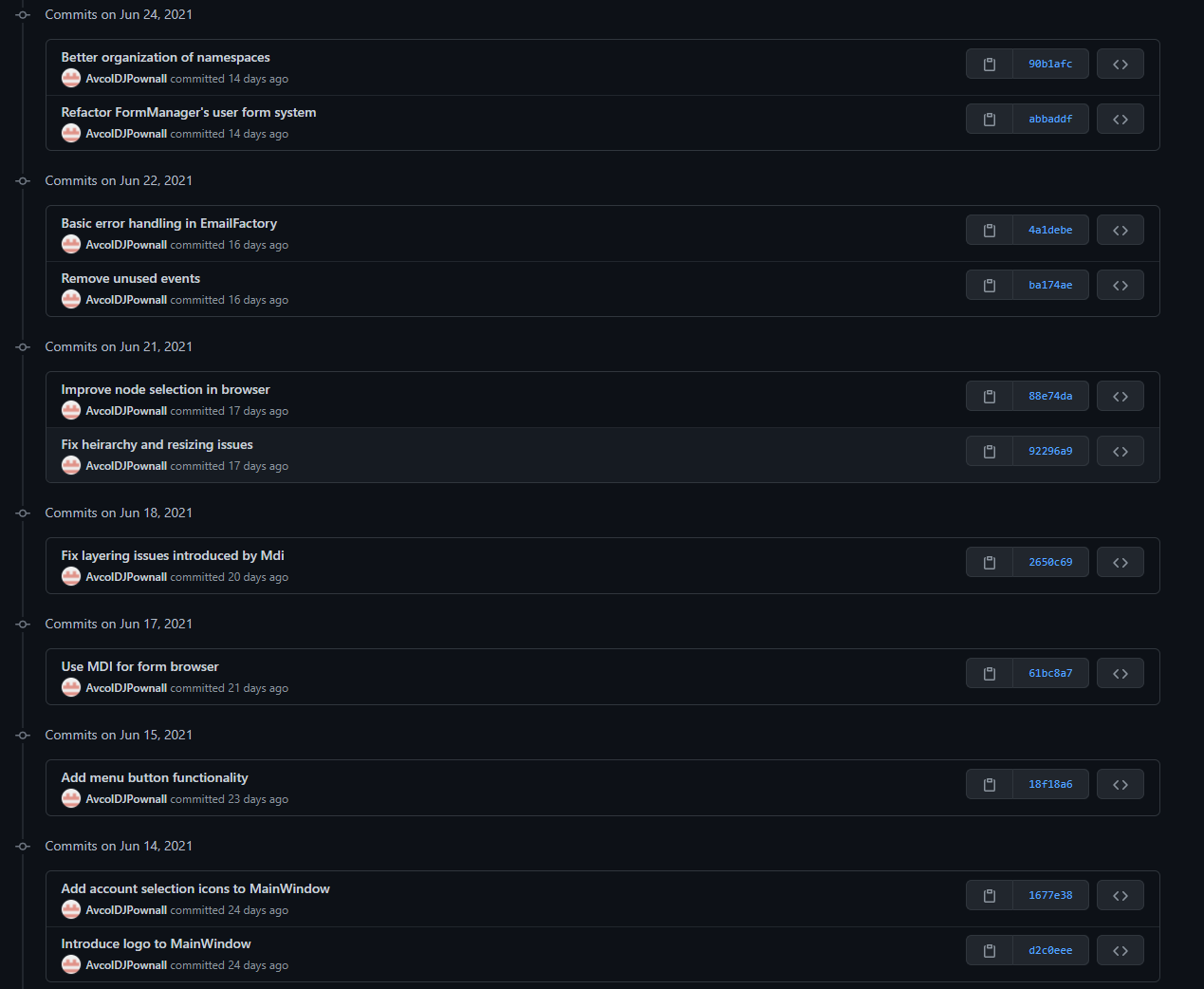
June 18 - The transition to Mdi caused numerous visual issues, so some modifications were needed.



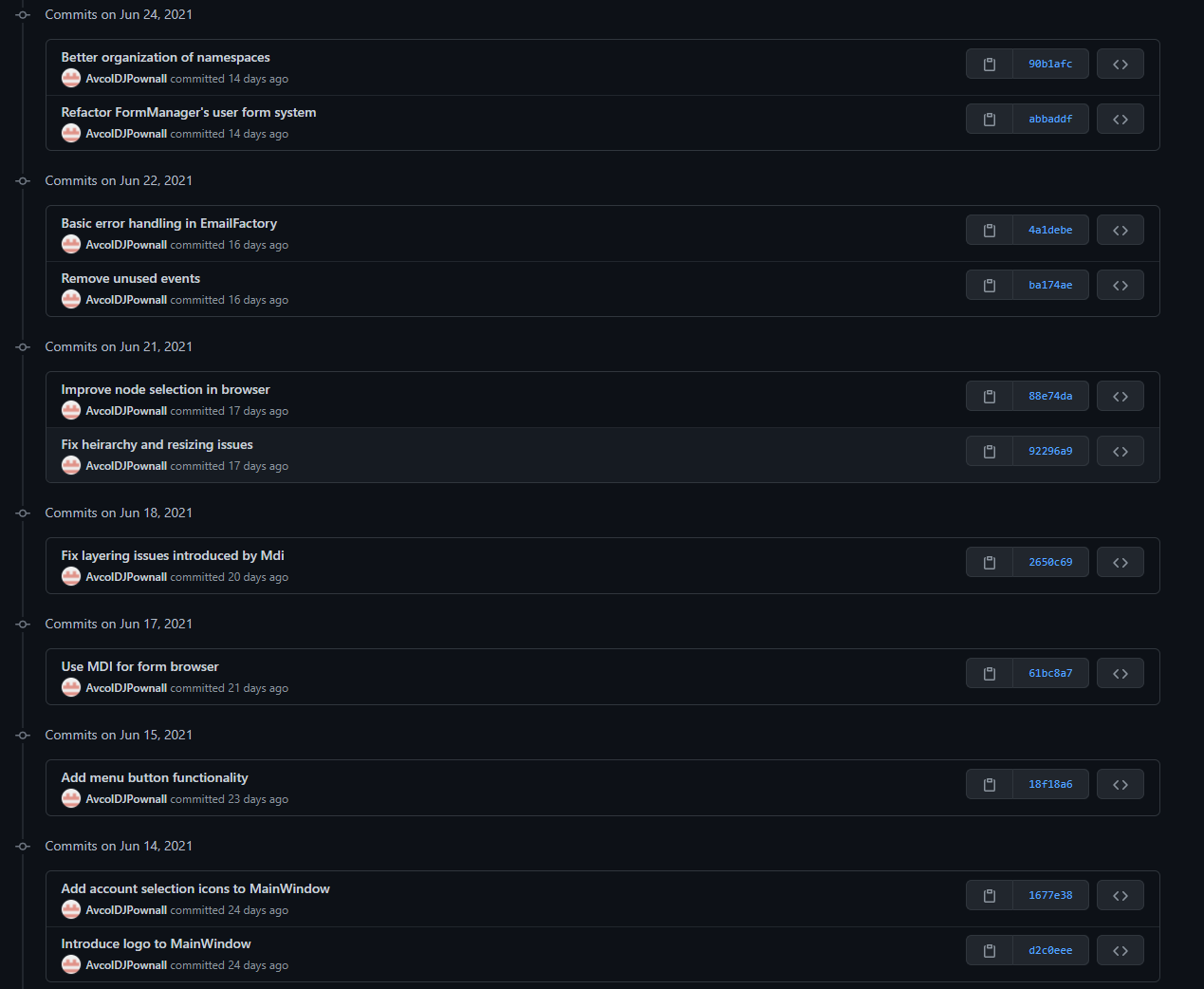
June 21 - Some additional fixes were needed to fix window resizing issues.



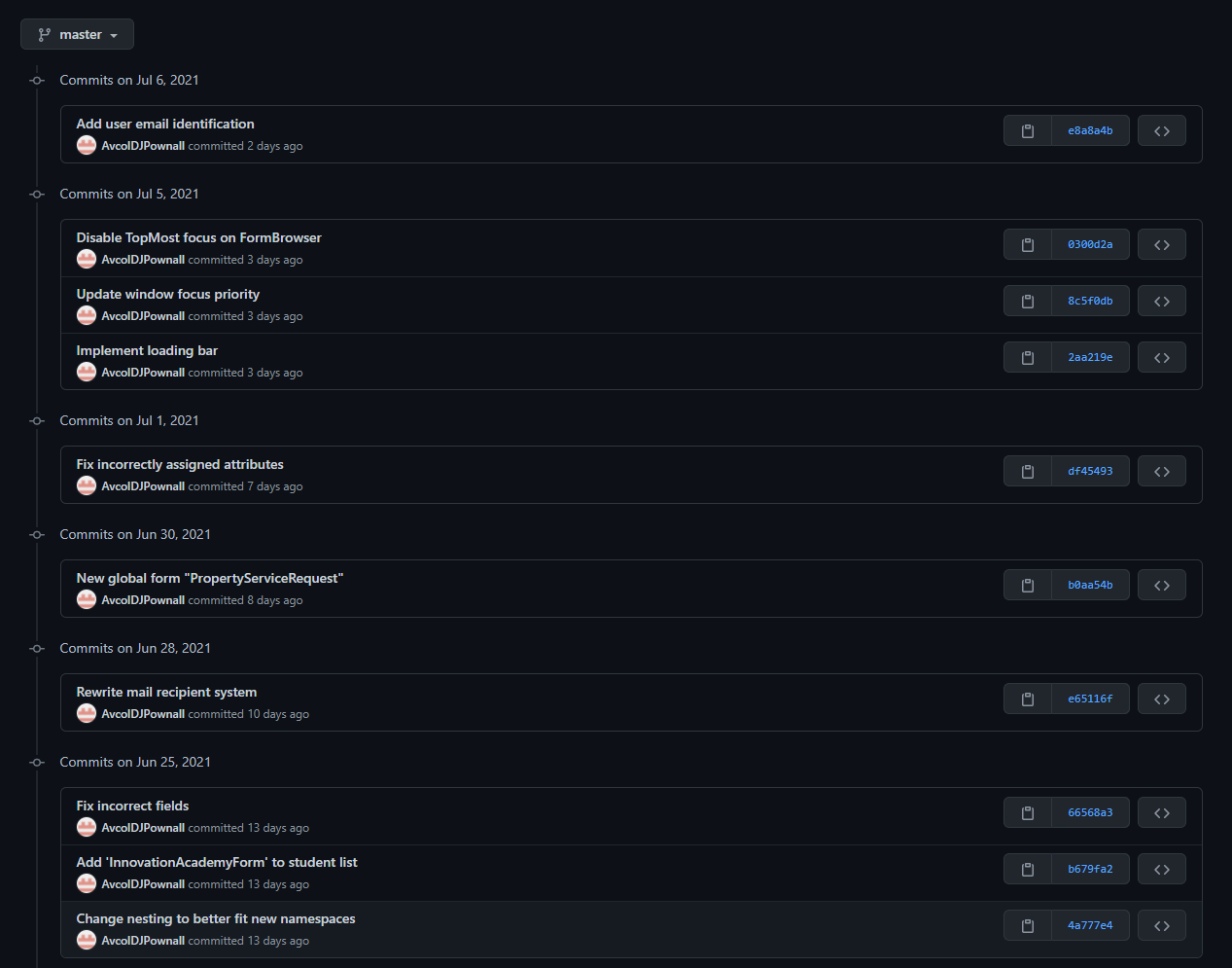
June 22 - Basic error messages were made to inform users if they were missing required fields.



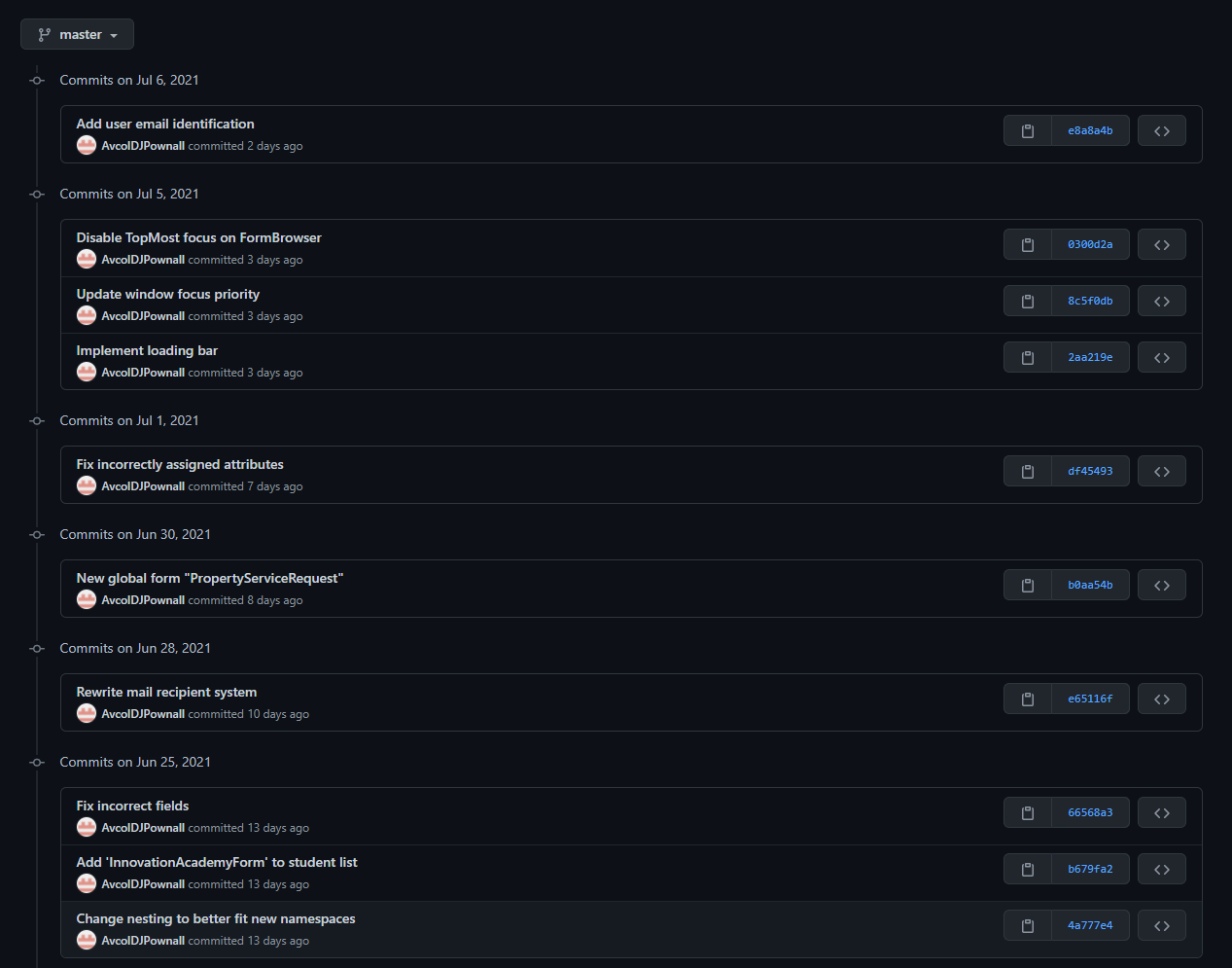
June 24 - This session was devoted to a project-wide clean up. A refactor of the FormManager class was also needed to make user types more modular.



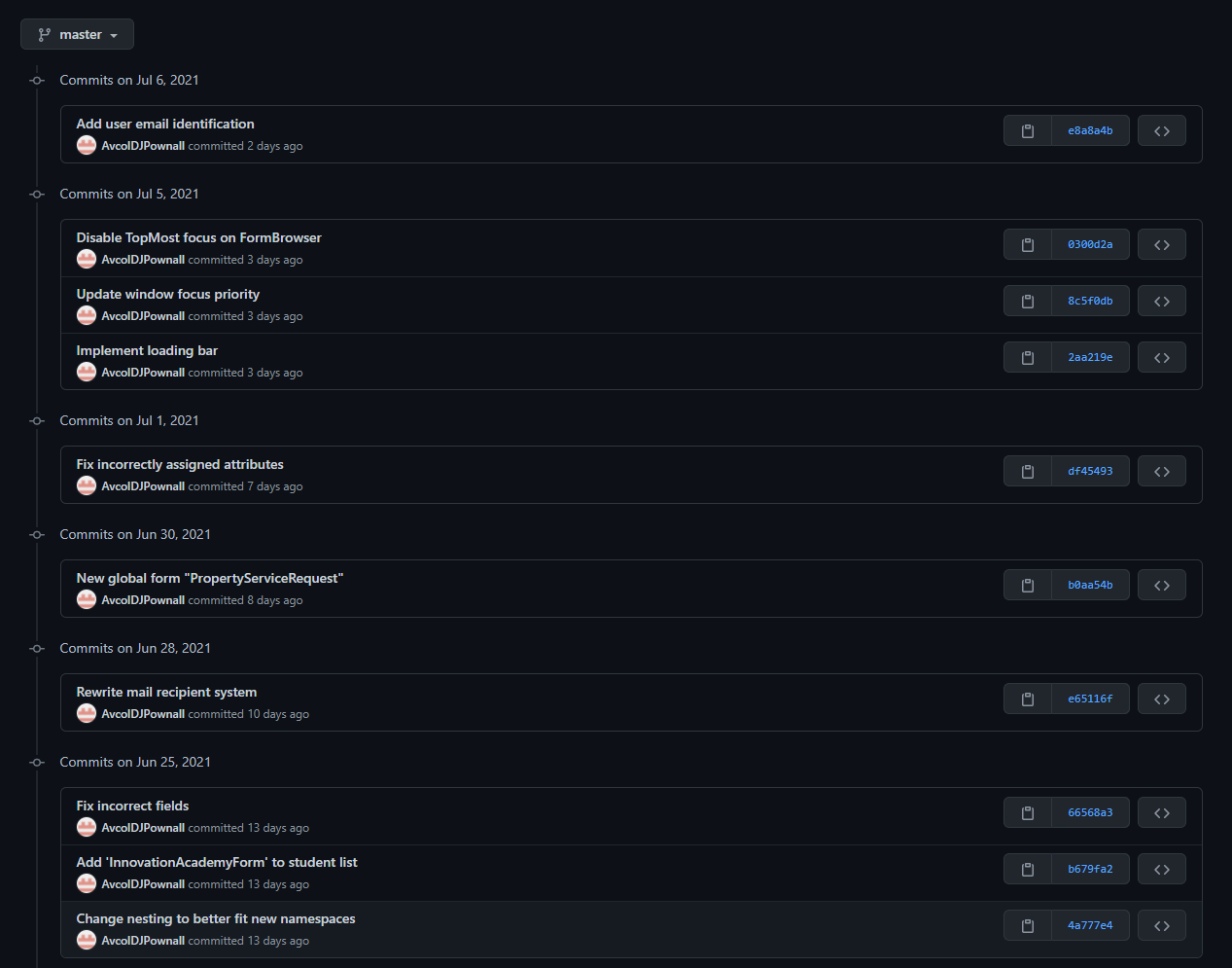
June 25 - A few changes were made, including some visual improvements and a new student form class.



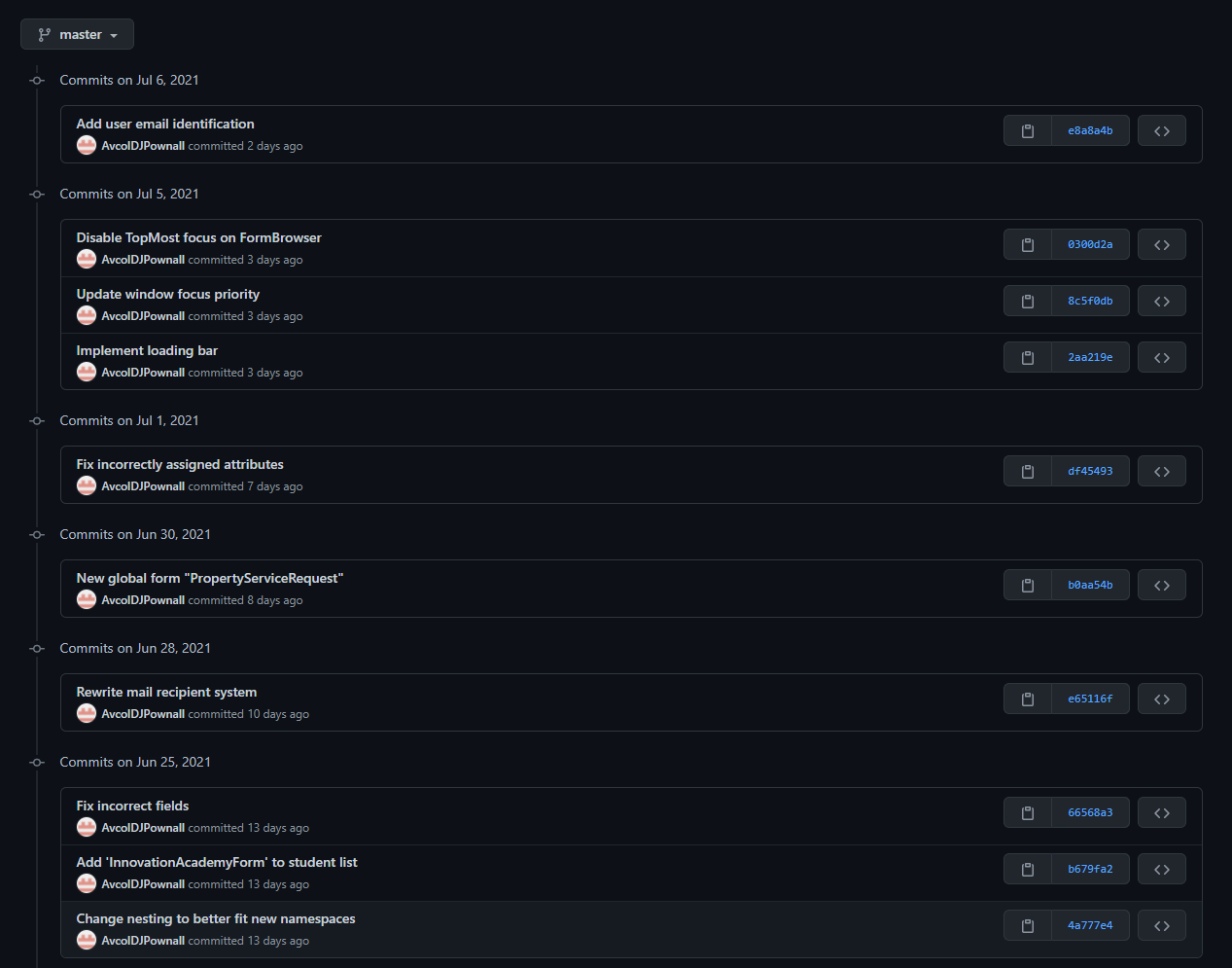
June 28 - A complete rewrite of the EmailFactory class was performed for better support for multiple recipients.



July 5 - A loading bar was implemented for clear visual feedback while uploading forms.



July 6 - Student ID / Teacher code logins were introduced into the form browser window.



**July 23 (Final day of software development) -** A basic overview of Avcol Forms was written for the GitHub repo, development was largely finalized for a release candidate.

------------ End of development cycle ------------

# **Maintenance**

Due to numerous factors, this program will need to receive frequent binary updates and maintenance following its initial implementation.

New form entries currently must be made using Visual Studio’s designer tool. As such, implementing more forms would require recompiling the code, then pushing the updated binaries across the school. Improvements could be made in future to make adding new entries easier, such as an update system or automatically generating elements by fetching a json file from a webserver. While such an approach would result in significantly less freedom in design, automatically adding new submissions to a json file would require no prior experience with Visual Studio’s form designer toolset. Another approach might involve a tag-based markup language such as XML. This would be a good balance between freedom of design and approachability for new users.

Depending on the client’s preference, the application might benefit from authorization prior to accessing student and teacher forms. The application could include a login page which compares a username and hashed password to a database entry. Though I did not include ths in the implementation due to security concerns, the school’s administrators could integrate such a system if desired.

The program would also benefit from a more robust email system, where a server-side application could handle form submissions. A future update could move authentication from the client application to an external server, greatly improving security. This would practically be a requirement for continued use in Avondale College. Decompiling AvcolForms using third-party tools such as Ghidra confirmed that an attacker could access the SMTP server’s credentials if they were to gain access to the binaries, potentially allowing them to impersonate the school’s trusted mail server.

Overall, Avcol Forms has achieved the basic levels of functionality which this project aimed to meet in its initial design stage. The final release candidate has proven that such an application would hold some degree of practicality in Avondale College. While a full implementation would involve integrating a variant of this application into an internal mail server, Avcol Forms could realistically be used in a school environment given these aforementioned modifications.

**References**

Tools used in AvcolForms security research/evaluation:

* [Ghidra (provided by the NSA)](https://ghidra-sre.org/)
* [ILSpy](https://github.com/icsharpcode/ILSpy)

Documentation referenced in development:

* [Windows Forms for .NET Framework 4.8](https://docs.microsoft.com/en-us/dotnet/desktop/winforms/windows-forms-overview?view=netframeworkdesktop-4.8)
* [Read/Write MemoryStream objects in .NET Framework](https://docs.microsoft.com/en-us/dotnet/api/system.io?view=netframework-4.7.2)
* [SmtpClient Overview for .NET Framework](https://docs.microsoft.com/en-us/dotnet/api/system.net.mail.smtpclient?view=netframework-4.7.2)