Unit 11

Web Development in Python (Reflection)

This week's unit focused on familiarising the students with a basic web app server and using Python to display data on a web page.

Interface Paradigm: MVC

Introducing the Model-View-Controller (MVC) pattern was an excellent way to present the basic building blocks of displaying information via web pages. The basic premise behind MVC is the *separation of concerns* between *what we see* and *how the system acts* on the data. This is, in contrast, to monolithic design patterns previously where presentation logic and business logic comingle in the same codebase.

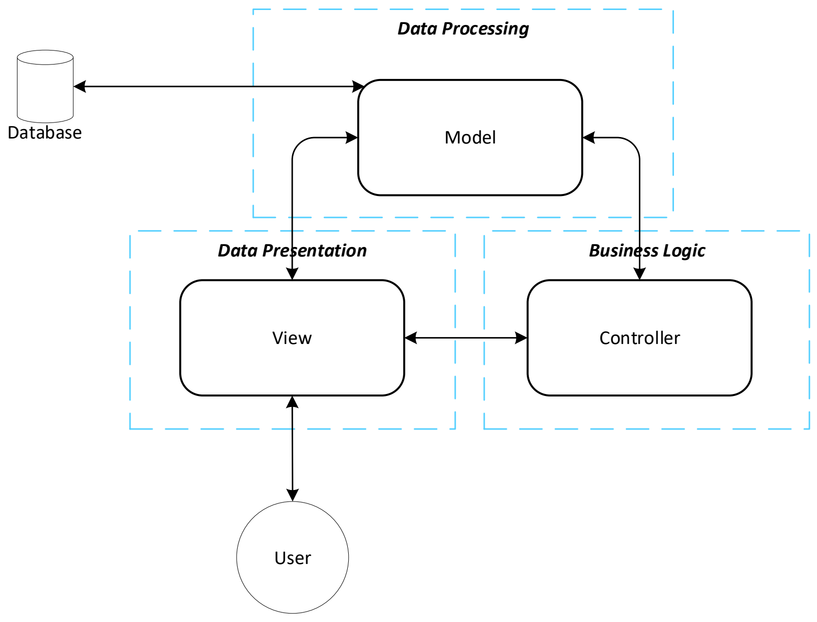

Figure 1 MVC pattern (created by author)

Here the ***Model*** holds information about business data. This data is presented to a user via a ***View.*** The user interacts with a ***Controller,*** which in turn updates the underlying model.

In monolithic systems, the situation looks as follows:

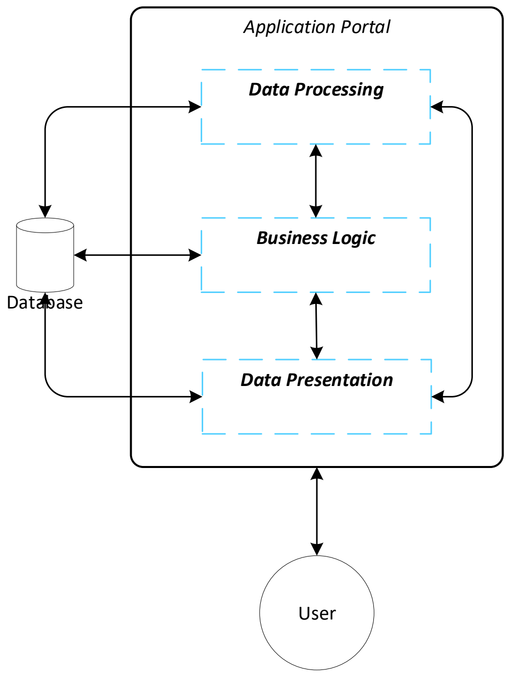

Figure 2 Monolithic interfaces (created by author)

The biggest issue with monolithic interface design is how tightly coupled together are data structures, business logic and presentation logic. Such tight coupling means that a bug introduced at any “layer” has a ripple effect throughout the system. Such costs tend to rise exponentially the longer it is left undetected (DeepSource, 2021). For this reason, isolating each component within an information system has become the preferred approach to developing user interfaces. This is because it is easier to isolate and correct bugs within a separate, isolated element than in an entire connected chain of components.

Before researching this topic, I was under the impression MVC is a recent invention. It was a pleasant surprise to discover that the MVC pattern was designed in 1979 by Trygve Reenskaug (Reenskraug, 1979). He states the reason for MVC’s existence was as “a general solution to the problem of users controlling a large and complex data set”. The paper is a fascinating read into the thinking behind models, views and controllers especially considering the invention of MVC was before modern web browsers took off.

Other Interface paradigms: MVP & MVVM

- ***Model-View-Presenter (MVP).*** A variant of MVC was introduced in the early 1990s. It achieves better decoupling between the domain and interface layers; the Presenter does not know the system. (Daoudi et al., 2019)

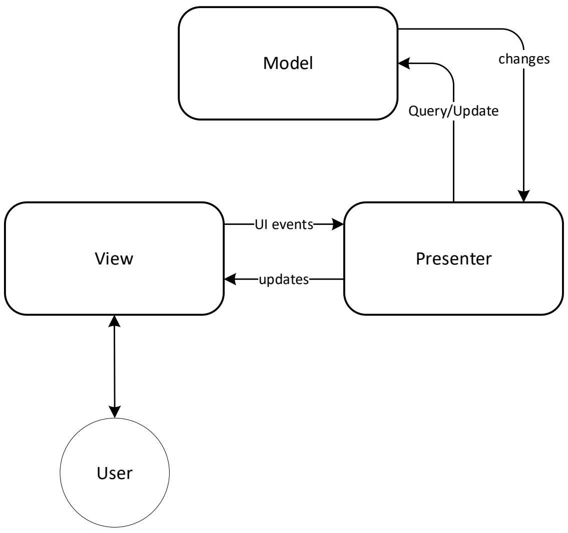

Figure 3 MVP pattern (created by author)

- ***Model View ViewModel (MVVM).*** This pattern is an evolution of the MVP pattern. The ViewModel is simple a model representation of a View. MVVM is the recommended design pattern for use on Android applications (Google, 2018).

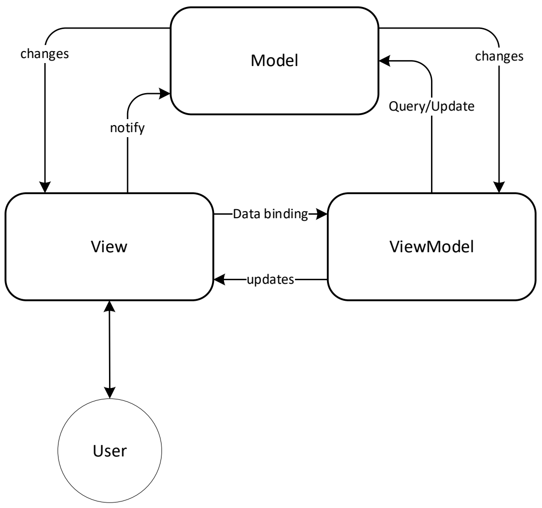

Figure 4 MVVM pattern (created by author)

Considering then the patterns available, Daoudi et al. (2019) note that 57% of Android apps they analysed (5480 in total) utilise the MVC pattern, while only 8% opt for the MVP pattern. I believe the results are a good reflection of the strength of the MVC pattern, namely, the separation of concerns.

Simple Web Development

Moving on from an understanding of MVC and the role it plays in web development. This unit then introduced the **Flask** library to host a simple web server written in Python. Flask is a micro framework written in Python, created by Armin Ronacher in 2004 (Pallets, 2010). Also introduced were Jinja templates that utilise special syntax within HTML pages to render content. Django inspired Jinja's templating system, which is described as the most used template engine for Python (Paletts2, ND)

So, this section of the unit matches the MVC concept. Flask fulfils the Controller aspect; Jinja templates meet the **View** aspect, and custom data provided to the templates complete the **Model** aspect.

Jinja templates are similar in concept to other web-based libraries, such as (Vue, ND); (Angular, 2020); (Blazor, 2021), and to a lesser degree, React (Facebook, 2021). However, React uses JavaScript eXtension (JSX), which allows returning HTML syntax within JavaScript code. I already had previous experience working with various templating techniques and found it easy to ease into the Python version of Jinja templates.

References

Angular (2020) AngularJS: Templates. Available from https://docs.angularjs.org/guide/templates [Accessed 22 Jul 2021].

Blazor (2021) ASP.NET Core Blazor templated components. Available from https://docs.microsoft.com/en-us/aspnet/core/blazor/components/templated-components?view=aspnetcore-5.0 [Accessed 22 Jul 2021].

Daoudi, A., ElBoussaidi, G., Moha, N. & Kpodjedo, S. (2019) An exploratory study of MVC-based architectural patterns in Android apps. *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*:1711-1720. DOI: https://doi.org/10.1145/3297280.3297447

Deepsource Corp. (2021) The exponential cost of fixing bugs. Available from https://deepsource.io/blog/exponential-cost-of-fixing-bugs/ [Accessed on 22 Jul 2021]

Facebook (2021) Introducing JSX. Available from https://reactjs.org/docs/introducing-jsx.html [Accessed 22 Jul 2021].

Google (2021). Guide to app architecture. Available from https://developer.android.com/topic/libraries/architecture/guide [Accessed 22 Jul 2021].

Paletts (2010) Flask web development, one drop at a time. Available from https://flask.palletsprojects.com/en/2.0.x/ [Accessed 22 Jul 2021].

Paletts (ND) Jinja. Available from https://www.palletsprojects.com/p/jinja/ [Accessed 22 Jul 2021].

Reenskaug, T.M.H. (1979) The original MVC reports.

Vue (ND) Template Syntax. Available from https://vuejs.org/v2/guide/syntax.html [Accessed 22 Jul 2021].