Colouring Cities Plugin Feature - June/July 2024 Report

By Gabriel C. Ullmann, Koa Wells and Samane Vazirian, Next-Generation Cities Institute

Introduction

Colouring Cities is an application "to support an international network of low maintenance open data platforms, managed by academia, providing standardised microspatial data on building stocks, at building level" (website). The colouring-core GitHub repository contains the main features of Colouring Cities, and each group forks the core to add their own features on top of it, adapted to their needs. While this allows flexibility for each group to go on with their business, this also causes each fork to drift away from the core architecture over time. For example:

- Each group may use different file, folder, component and function naming conventions.
- Each group uses different text strings (for different languages).
- Implementations may be created using different paradigms (MVC, MVVM, MVW).

As a result, over time it becomes harder and harder to integrate changes from the fork into the core repository, because merge conflict increase in frequency, and sometimes these conflicts are unresolvable due to divergent implementation strategies. To mitigate this problem, we propose the development of a plugin feature, through which we could keep a stable core and let groups add, change and remove functionality "around" the core at ease. However, implementing such a plugin feature into an already implemented application presents several challenges, such as:

- We do not know which features can (or should) become plugins
- It is not immediately clear which group of files and folders represents one feature
- The very existence of a plugin feature implies several subsystems which need to be developed, such as:
  - Plugin detection, installation and un-installation
  - Plugin version control and dependency resolution (i.e., which plugins do my plugin need to work?)
  - The creation of a Core API through which plugins can "talk" to the core

Research Questions

To create a minimum viable plugin feature for Colouring Cities, we must first understand Colouring Cities's architecture and extract elements from it that we want to "pluginise". We decided to start with the "Energy Performance" (a.k.a Sustainability) feature. To pluginise it, we must answer the following research questions.

- **RQ1:** Which files and folders represent the Energy Performance feature within colouring-core?
- **RQ2:** Can we extract these files and folders to an external module/package?
- **RQ3:** Once extracted, can we make the "Energy Performance" feature "plug into" the core via some kind of mechanism (e.g., importing)?

Method

First of all, we decided to search for academic literature references on how to create a plugin feature. One of the few descriptions of this kind of architecture we found were the Microkernel Architecture described by Richards (2022) and the Plugin pattern by Fowler (2002). While Richards describes plugins as "standalone, independent components", he does not provide a concrete implementation of the feature. Fowler, on the other hand, suggests the use of interfaces and inheritance.

Moreover, we looked for concrete examples of applications with plugin features so we could reflect on how they implement these features and perhaps use them as inspiration for our own. Google Chrome, Unreal Engine and Unity are some examples which were on the top of our minds. They all assume the responsibilities we mentioned in the introduction, such as installation, dependency resolution, etc. However, in these applications the plugin system has existed since the first version, which is not the case for Colouring Cities. Therefore, we observe that, while a plugin feature is not something unprecedented, implementing it within a structured architecture proves to be challenging, and is not a common occurrence in commercial software. To understand the high-level architecture, we modelled and visualised Colouring Cities in Moose, a software analysis platform based on the Pharo language. Its Architectural Map (Figure 1) provides us with an overview of the division of responsibilities within the application. For example, we can see the application clearly separates visualisation from data handling by creating structures such as controllers, routes and data access scripts. There are also other supporting architectural structures such as error handling and config files.

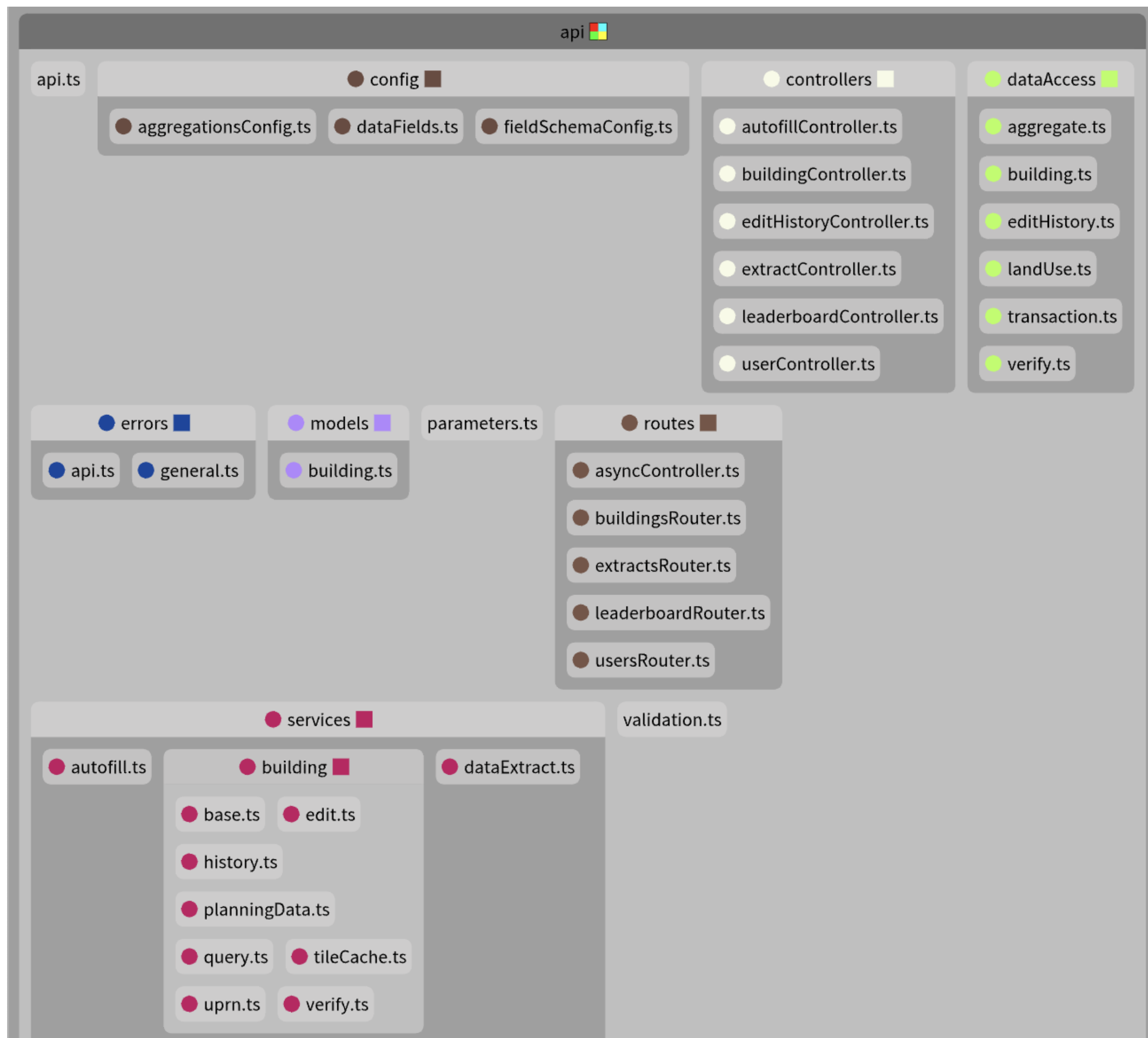

Figure 1 – Excerpt of the "api" folder of colouring-core

Once we understood the main architectural points, we moved on to more specific implementation aspects, such as how React components are built. During this step, we asked Mateusz for help, and he explained some aspects to us which were not immediately evident by simply reading the code, such as how the routing system works. Based on our conversation with him and our code exploration, we produced a high-level diagram describing the flow of data in Colouring Cities components (Figure 2).

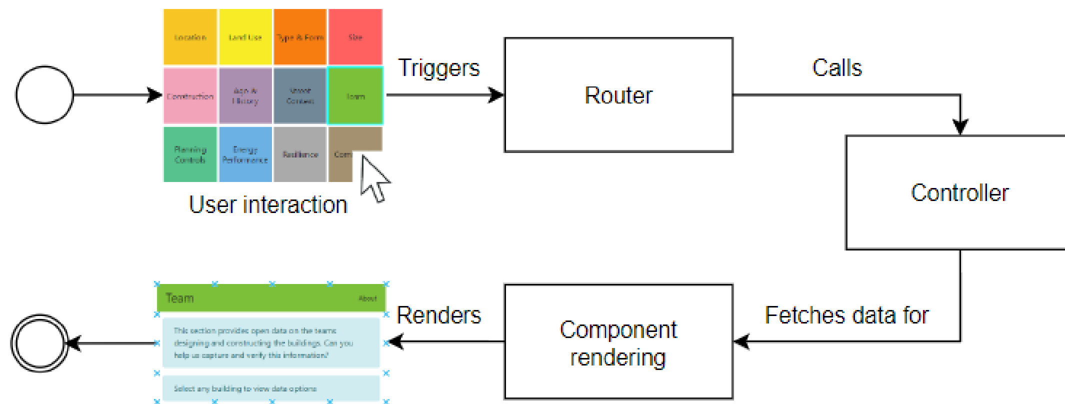

Figure 2 – High-level view of the front-end architecture and interactions

Results

So far, we managed to understand the overall purpose and concept of Colouring Cities and how its main features connect to React and the back-end. However, we did not manage to answer RQ1 merely by using static analysis. Therefore, we intend on running the application and experimenting with live changes to understand how "Energy Performance" relates to its surrounding components, where the entirety of tis code is located and how we could extract it. Answering RQ1 will help us, consequently, to answer RQ2 and RQ3 too.

Conclusion

We identified a potential architectural challenge for Colouring Cities: architectural drift caused by individual groups forking colouring-core to develop unique functionalities. To mitigate this issue, we proposed the development of a plugin feature that allows for the stable core to be maintained while enabling flexibility in adding, changing, or removing functionalities. Our initial focus on "pluginising” the "Energy Performance" feature serves as a proof of concept for this approach. In the coming months, we will work to identify, extract, isolate and reconnect this feature with the core.