Riley Blair 300371586

Remote LED Customisation & Control

Chromolite Architecture Changes

Chromolite's current architecture supports a universal desktop interface that can be used on most desktop platforms. With the aim of being a universal application for unifying the customisation and control of RGB LED devices, a more suitable and more applicable architectural solution is to provide the end-user with a web interface that can be used on any web-enabled device to access a REST API that controls the connected IoT devices. By providing this architecture, Chromolite would enable improved remotely operated customisation and control of RGB LED devices.

Proposal of Architectural Changes

Chromolite could benefit greatly from providing users with a web-based interface for the remote control and customisation of RGB LED devices as this would be a significant step forward in providing universal control for all users. In order to provide this service, two fundamental changes are required to the existing Chromolite architecture. Firstly, a web interface would need to be developed which supports HTTP POST requests to a active web server. Secondly, the web server would need to provide a service that the interface can communicate with, likely through the use of a RESTful API (Representational State Transfer). This proposed architecture would support the existing architecture, as it reflects the client-server architectural model currently in use within Chromolite.

The first of these two significant changes to the architecture is the web interface. Currently Chromolite supports a desktop and Android application, but for users who do not use either of these platforms, they are unable to use the software due to compatibility constraints. To solve this issue and make Chromolite universally accessible across all internet-enabled devices, the most appropriate solution would be to develop a web base interface. This interface would be cross platform and would simply require a web server in order to run. It could then communicate with the connected IoT devices by sending HTTP POST requests to a specific port at a specific IP address and letting an application running there handle the processing of the requests.

The architecture behind this web interface would likely be very simple to set up, consisting of a web app based within the Angular5 framework due to its extensibility and abstraction of complex components of the web architecture. By utilising this type of architectural framework, a highly dynamic interface can be established that can grow and scale as necessary with the size of the overall project with minimal changes required to support rapid growth. This would ultimately put the Chromolite application in a position where it could be distributed to a much wider audience with a significantly larger user-base. In providing this extension to the existing interface architecture, the Chromolite application would have greater applicability and would be more aligned with the creator's goals of creation universal and unified control of RGB LED devices.

The second component involved in this architectural change would be a RESTful API that can receive requests wirelessly from the web app and process these into commands for the connected IoT devices. This architectural change is largely to support the aforementioned architectural change to the application's interfaces, but will also serve as a new means of control for the system. Essentially, the REST API would be hosted on a local web server that is run in parallel with the desktop application.

In order to achieve the desired REST API and web-server functionality, a simple Java and Maven application could be established using the Spring Maven Plugin. This would provide a means to rapidly develop a REST API that could be run locally or hosted remotely on a managed web-server, with high extensibility and scale built into the core of the REST API's framework. This would primarily be achieved using a Maven web-server built in Bash and Shell scripts, with a Java application managing the API endpoints and their desired functionality. The use of query strings would further enable more granular customisation options with real-time feedback from the output devices. Spring's framework enables a level of complexity abstraction that will make the initial and continued API development seamless with the rest of Chromolite's fundamental architecture.

By implementing these two significant changes to Chromolite's architecture, the stakeholders involved are benefitted by having greater universal applicability, and are also closer to Chromolite's vision of universal unification of remote RGB LED control. Stakeholders would further benefit by having a system that is more scalable than the existing architecture, with the ability to reach and support a greater number of users who in turn, would become stakeholders of the project. While these proposed architectural changes do apply to the fundamental architecture of Chromolite, they do not restrict existing functionality, and instead simply provide greater applicability and extensibility to the existing desktop architecture.

Developing a RESTful API

The RESTful API provides the largest architectural change to the project out of the two architectural changes implemented. This new architecture exists within the desktop application and operates as a separate web-server specifically for receiving HTTP requests at a specific port of the host's IP address. Upon having one of the API endpoints hit by a request, the API passes off execution of the desired function to the appropriate classes, and as such does not present a large functional change, but instead a architectural change in the way users can interact with and execute functional aspects of Chromolite. The REST API I developed for Chromolite is based within a Maven project with XML handing the project's structure and Java providing the endpoints and execrable functionality. All of this is compiled and run by Bash and Shell scripts which initialise and run a local server that accepts HTTP requests.

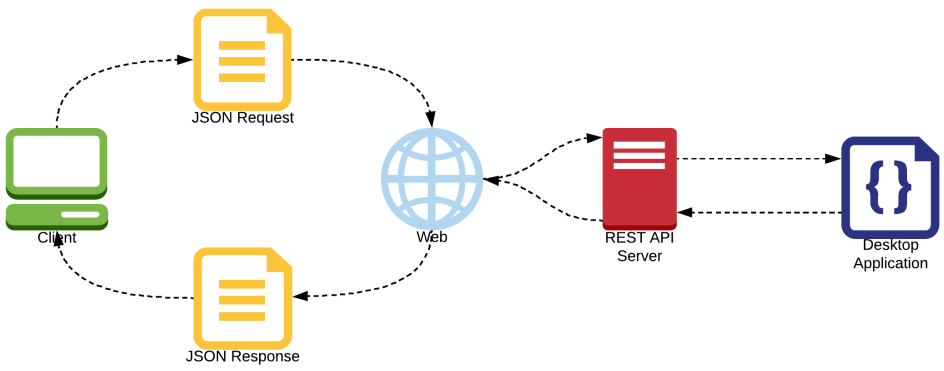
The Spring framework and Maven plugin was chosen for this because of their applicability and extensibility to Chromolite's existing architecture. By using them, a significant change to the existing architecture was not required, as they allowed for the REST API to be added without changing the current desktop server architecture. Because this is an API, it is simply to provide an interface between the existing functionality of the desktop application and any external client that wishes to communicate using HTTP, including the web interface outlined on the next page. While not changing major functional aspect of the Chromolite application, the REST API does provide a significant extension and improvement to the overall system architecture, allowing Chromolite to be controlled remotely from any device that can send HTTP requests.

When executing the Chromolite application on the desktop environment, another server is now initialised and run on a different port to the server managing communications from the Android app. By having REST API server on a different port, the communication channels can be kept separate, which also allows for greater modularity in the incoming communications assisting with future development of Chromolite. Having this running as a separate entity means that development can be focussed on any of the forward-facing interfaces and respective APIs with minimal impact to the others. Further to this, because the API is simply an interface to bridge the gap between a web-based implementation of Chromolite and the Arduino micro-controller, the functionality of the actual application can be changed independently of the API without the need to redevelop large sections of either.

A further Gradle version of the REST API was set up to ensure greater applicability of the changed system architecture. While at this stage the API is made to run on a local machine which can, in most cases, have the ability to install and run both Maven or Gradle, larger scale deployments or implementations of Chromolite in the future may have specific requirements that restrict the usable frameworks. By using both Maven and Gradle to provide two different means to execute and run the REST API server, Chromolite is now more applicable in more situations that it would have been without both options available.

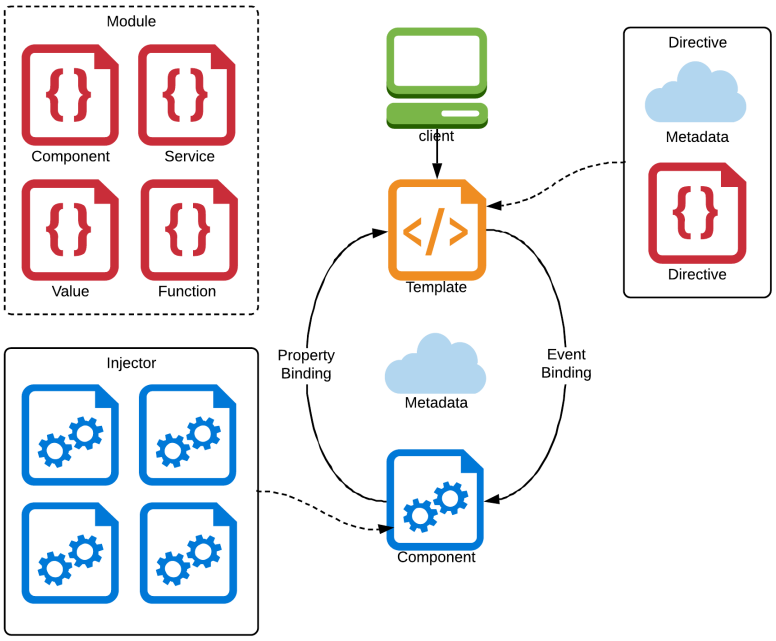
The architecture of the REST API itself is based upon Microsoft's REST API design guidelines and is abstracted using the Spring framework for Maven in order to reduce the complexity and make future development easier for someone who is not as familiar with REST API design or implementation. While the returned JSON body is generally not used in the case of a HTTP POST request, it does help in providing more information about the endpoint and the functionality that is carried out by hitting this endpoint. Requests to the API can be made either using URL extensions and query strings/parameters, or by sending a JSON payload to the endpoint. If the request is formatted correctly, then the API will trigger the desired function within the Chromolite application based on the queried parameters. If the request is incorrectly formatted, then an error response is returned instead for the requester to handle as appropriate.

The addition of the REST API and local web-server to the overall architecture of the project will provide significant benefits to Chromolite's stakeholders. The extensibility of the Spring framework used for the API will enable future open source developers of Chromolite to easily extend and modify endpoints, return headers and return bodies with no impact to the functionality of the actual Chromolite application. End-users of Chromolite with further benefit from the ability to have a universal web-based interface that functions using the REST API.



Extending Chromolite with Web Architecture

As previously mentioned, the web architecture developed within the Angular5 framework would enable greater usability and extensibility of Chromolite, taking it a significant step forward towards the application's goal of universal applicability. Angular5 was chosen as the framework for providing the web interface due to my familiarity with it, allowing for rapid development of a fully functional web interface within a day. This architecture also allows for near-seamless integration with the existing functionality with minimal changes required.



Angular's architecture integrates well with the architectural styles of the other interfaces/ applications currently within Chromolite. All components are sectioned into blocks, called modules, and provide a compilation context for components. All modules are compiled into functional sets in order to carry out the necessary functionality while leaving the HTML/ CSS template and TypeScript Component fairly well abstracted from module complexities. A simplified version of the Angular architecture used in the construction of this interface is shown to the right. At a high level, components define views, which are a set of elements visible to the end-user. Angular can modify these according to the program's logic and data through property bindings. Components also use services, providing specific functionality that is not directly related to the view. These services can be injected as dependencies to a component. Components and services both follow the decorator software design pattern and reflect this in their abstracted interactions with the other aspects of the overall architecture.

When defining the components for Chromolite, I analysed the major functions of the view and controller aspects of the MVC design pattern present in the desktop application and generated three components from this. The dashboard component houses much of the functionality and is responsible for directing requests to the aforementioned REST API via an injected HTTP client module. Further to this, an options component was defined for each of the three types of IoT devices currently supported by Chromolite. In doing so, the architecture is flexible and can scale as necessary if or when new device support is added. Decreasing coupling in this manner through Angular's built-in routing and component structure is also one of the reasons why an Angular-based web app is a good architectural choice for this interface.

Each aspect of the view and controller of the previous Chromolite architecture has been remapped to reflect the more scalable architecture of Angular's framework. By abstracting aspects such as event listeners and event handlers into Angular's event bindings and module functions, future open source developers of the Chromolite project will be able to compile simple to execute functions and click events with minimal knowledge of the inner-workings of the handlers. Because the open source community and open source developers are key stake holders in this project, they benefit from being able to easily add, modify, or remove functionality with minimal to no effect on other aspects of the application due to the low coupling and high cohesion of the Angular components. Changing Chromolite's interface architecture to reflect the architecture of an Angular-based web-app ultimately reduces the time any future developer will need to spend making both functional and non-functional changes. By having a well established and proven framework perform core actions and responsibilities rather than user-generated code that may include bugs, the reliability of the overall system and its underlying architecture are greatly improved.

The Angular web app, while not currently hosted on a managed web-server, can be executed locally with a single command and provide the user with a locally hosted interface to connect to the REST API. Because this is hosted locally, anyone with the correct IP address and port can also access a new instance of the interface. If put into production, a small number of dependencies are automatically changed for a managed hosting environment, allowing for Chromolite's interface to be distributed through a globally accessible URL and domain. Therefore, with this architectural change, Chromolite is able to reach a greater number of users, as well as adapt significantly faster to changes to the functional and non-functional aspects of the scope of the project.

Architectural Evaluation

The architectural changes implemented through the development of a web app and interface and RESTful API with accompanying web-server will provide a significant benefit to project stakeholders and bring Chromolite close to its goal of being a universal application for the unification, control, and customisation of RGB LEDs. After performing an analysis of the existing project's architecture it was clear that significant development and changes to the architecture would be required in order to make Chromolite a universal application that is supported across all platforms without needing to make significant changes to each application type every time a functional or non-functional aspect of the software is change. Therefore, it seemed appropriate to develop a web-based interface and a REST API in order to connect this with the embedded system as functional and non-functional components of the system could be updated, added, or removed in one place and be available immediately to all users. This is opposed to relying on end-users to update their application versions, and for open source developers to implement their changes on all platforms.

Open source developers, including the creator of the project, some of the key stakeholders in the project, benefit greatly by the more scalable and extensible interface provided by the Angular5 web app. Changes can be implemented at a greater rate due to the abstraction of common tasks such as event listeners and handlers, that would usually require large amounts of complex code in order to implement correctly. As previously mentioned, in the current architecture, a change made on one platform (desktop or Android) would need to be translated to the other platform prior to deployment to ensure consistency. The architectural change of utilising the universal applicability of the web addresses this concern, as changes to the project only need ti be made in one okay rather then in multiple places within the code base.

End-users are also at a great advantage from the architectural changes made. This is because the Chromolite application is now universally accessible from any internet-enabled device. This is key to assisting Chromolite to achieve its goals of being a universal application for remote RGB LED control. Users can potentially have greater access to the application through a hosted web application (if approved by the project's creator) as they are not limited to specific devices, or having to be within a certain range of the Arduino micro-controller.

The diagram below outlines the new project architecture at the high level, with the web application providing a new interface for Chromolite users and the REST API and REST API web-server running locally within the desktop application. By designing the architecture with a locally hosted web-server running the REST API, the project now has the potential to be extended to run on a managed hosting service externally from the connected IoT devices. Due to the way the new architecture has been developed, the API server and web client could be completely separated from the existing architecture and instead be based solely in the could, reducing the hardware requirements and becoming the primary architecture of the Chromolite project.