Jurny/Janus Implementation details

# Introduction

I am writing this document to capture the implementation details of the entire Jurny system design. There is no organization or formal structure to recording these details. The details are recorded for the purposes of understanding historical architectural decisions.

# Initial system architecture

## Terminology

## Renderer

A renderer is a module that is used to generate content from a given input. The given input could be a set of parameters or a JSON object itself. The renderer module can have a layer of code that interprets different types of modules from JSON to XML to binary. Behind the renderer interface, we have functions that absorb data in the form of input parameters and generate the required HTML.

## Renderer types

There are multiple renderers to be designed. First, we need to design a renderer to render a particular type of object. For e. g. the base object renderer which concerns itself with the shape, size and color of each activity object.

The activity objects themselves occupy the space of another parent day object. A renderer has to be designed for the day object as well. An itinerary object will render all the days and the connected information with the itinerary. There needs to be a renderer for the original itinerary object too.

The rendering has to be done hierarchically and perhaps recursively as the input format of the itinerary object is traversed. Rendering should also offer different configuration which means there needs to be different views which the renderer has to offer that can be suited for the needs of the users of the application. Rendering configurations/views should be scalable in that adding a new view or a rendering configuration should be easy for the developer.

For now, we will just hard code the renderer.

What are the rendering tasks?

We first need to define how the entire itinerary would look, how each day would look in the itinerary and how each activity will look on each day. There needs to be a top level function like this:

Renderer1.render(itineraryObj);

Each renderer attaches itself to an itinerary object interpreter which reads the itinerary object and offers functions such as:

* GetNumDays() : Get the number of days in the itinerary
* GetAllActivitiesForDday():
* GetActivity()