# Assignment 4

## Custom UI Dashboard

Student Numbers:
A0155858M, A0154776R

Student Names:
Adil Bin Azmoon, Lim Yong Song

# UI Design Improvement 1

## Visual hierarchy (Balance)

### Instructions:

Reorganise the layout after increasing the size of the parameter control component and temperature component while reducing the size of the floorplan component

### Benefits:

There are 3 main components in the design, time-series graph, floorplan and parameter control (start/end date with samples size selection).

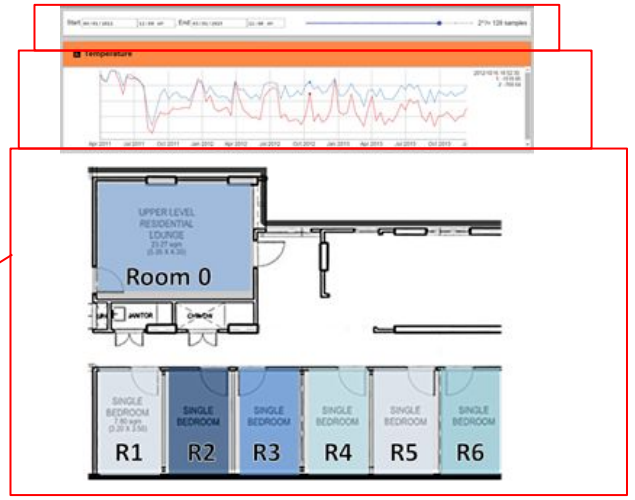
Before the UI change, the user may not see the parameter control component, which is the most important component for the user.

The attention is instead drawn to floorplan due to the imbalance.

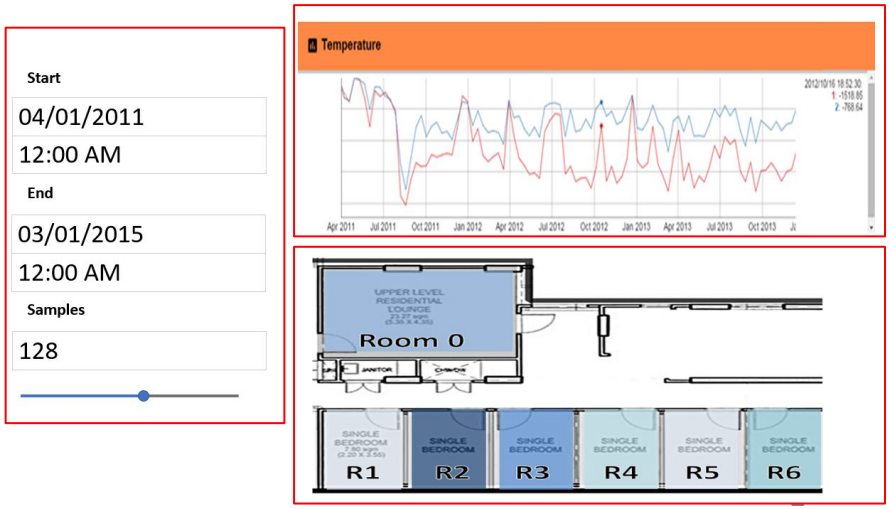
To achieve better balance of visual attention, the components are resized and relocated so that equal attention can be drawn to the graphical components along with the parameter controls.

After this use of positioning and control of size, an asymmetrical balance is achieved that allows the attention to be shared over equally all the components.

**Before**




**After**



# UI Design Improvement 2

## Grouping (Color)

### Instructions:

Use analogous colors for header and background to highlight related components and complementary colors to show differences.

### Benefits:

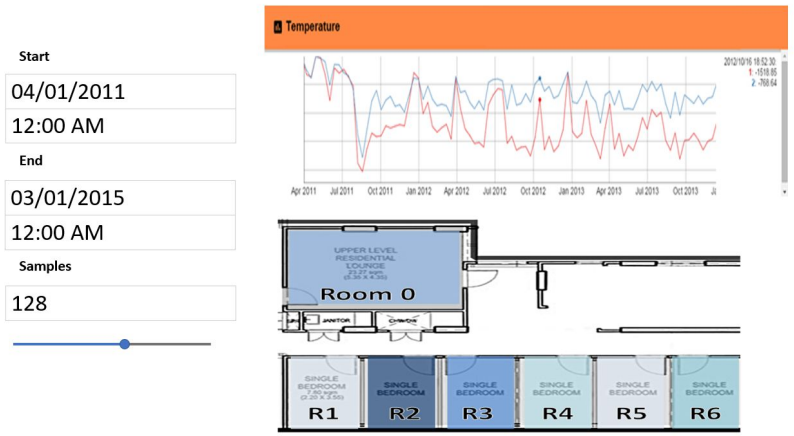
The parameter control (start/end date with samples size selection) component is mainly used to control the bulk of what is displayed by the temperature and floorplan components, while changes in temperature component (when it is zoomed to adjust the time period) and floorplan components (when certain rooms are clicked) results in minor changes in the others.

Before the UI change, it is hard to differentiate each component between one other and they seem to be clumped together as a whole group.

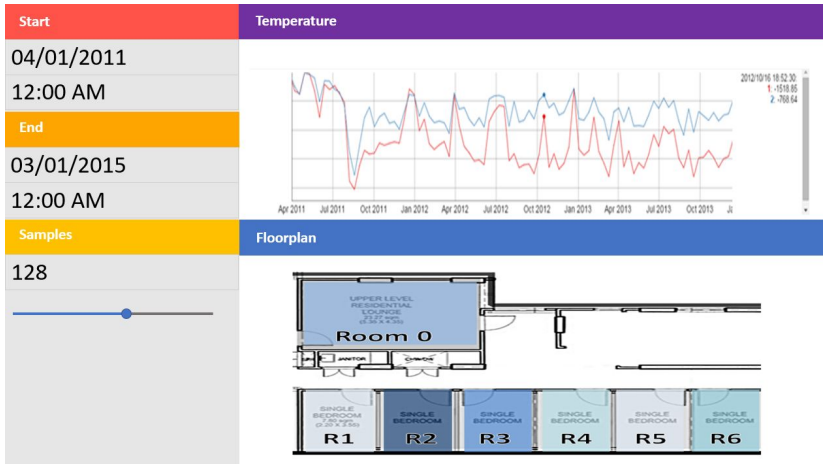
To help the users differentiate the different groups more easily, complementary and analogous colors are used, with parameter control components being more orange and the graphical components (temperature and floorplan) being more blue.

After the proper use of colors, it can be seen that it is a lot easily to differentiate between the different groupings and create mental shortcuts for the user to navigate the page.

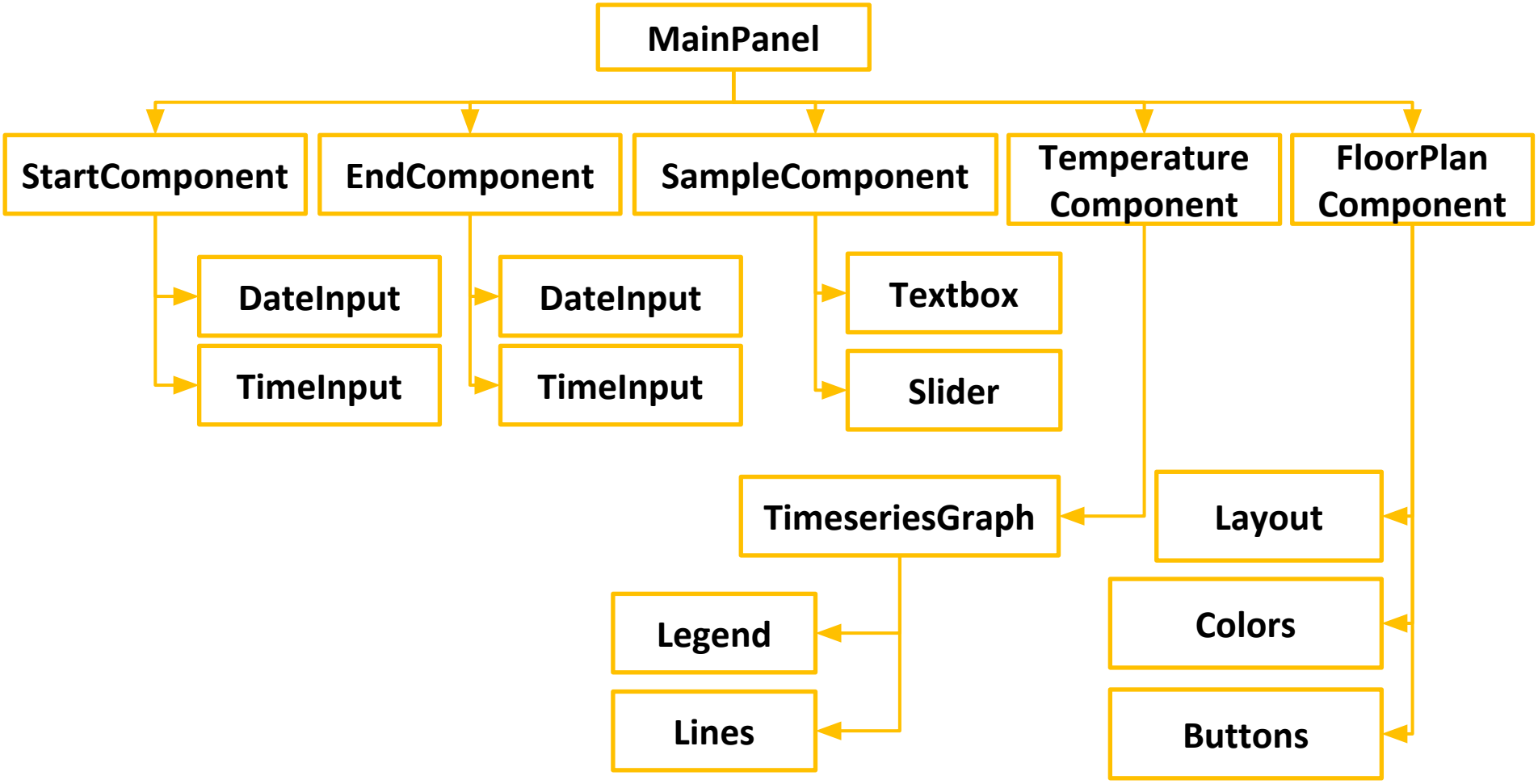
**Before**



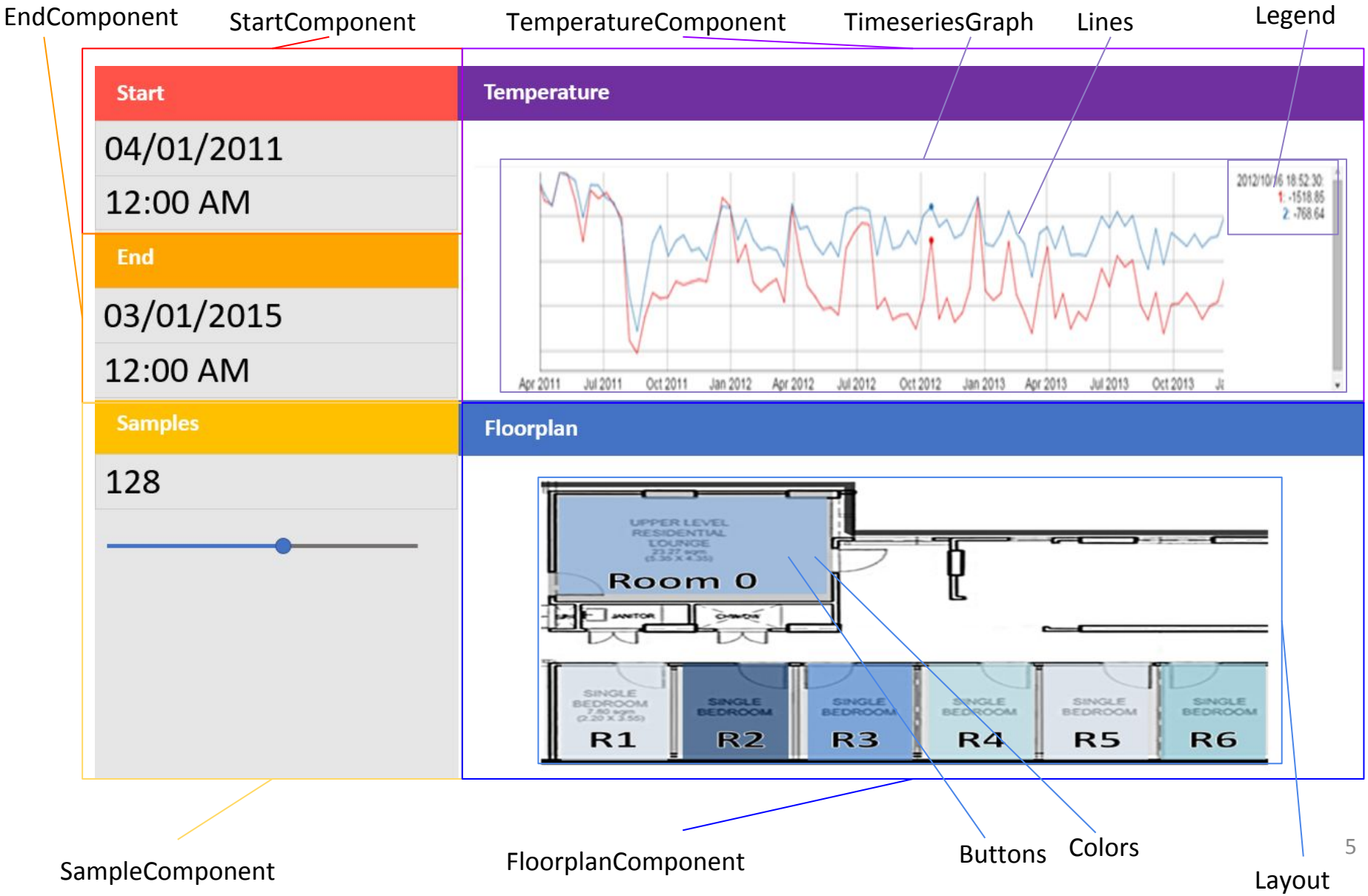

**After**



# View Tree

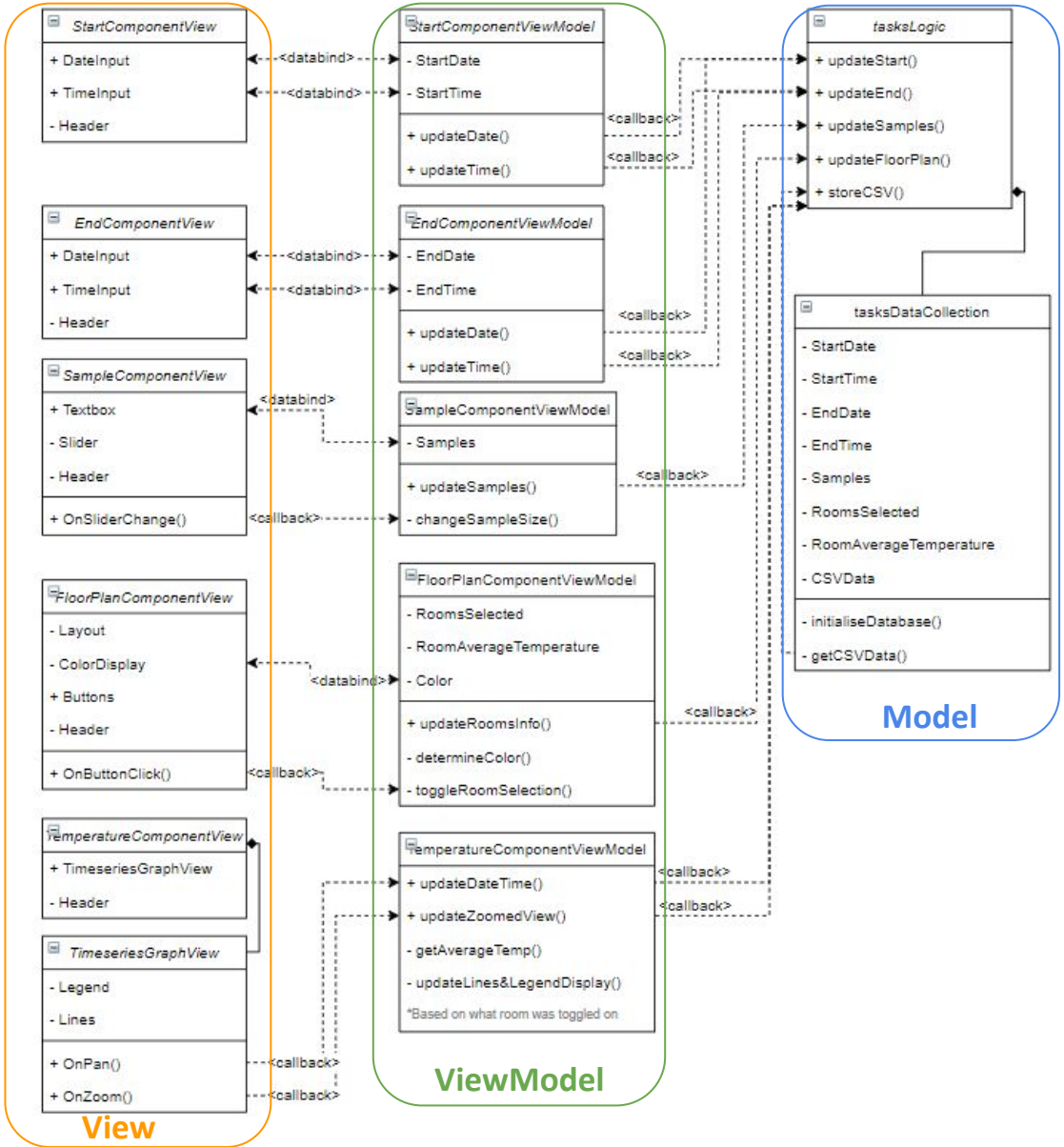

MainPanel
StartComponent
EndComponent
SampleComponent
Temperature Component
FloorPlan Component
DateInput
TimeInput
DateInput
TimeInput
Textbox
Slider
TimeseriesGraph
Layout
Legend
Lines
Colors
Buttons

# UI Components



# UML Class Diagram (with Objects)

https://app.diagrams.net?lightbox=1&highlight=0000ff&edit=_blank&layers=1&nav=1#G1uI9p1c9fHfEg6FRmU5kLrztBST-DSUHc




Notes:

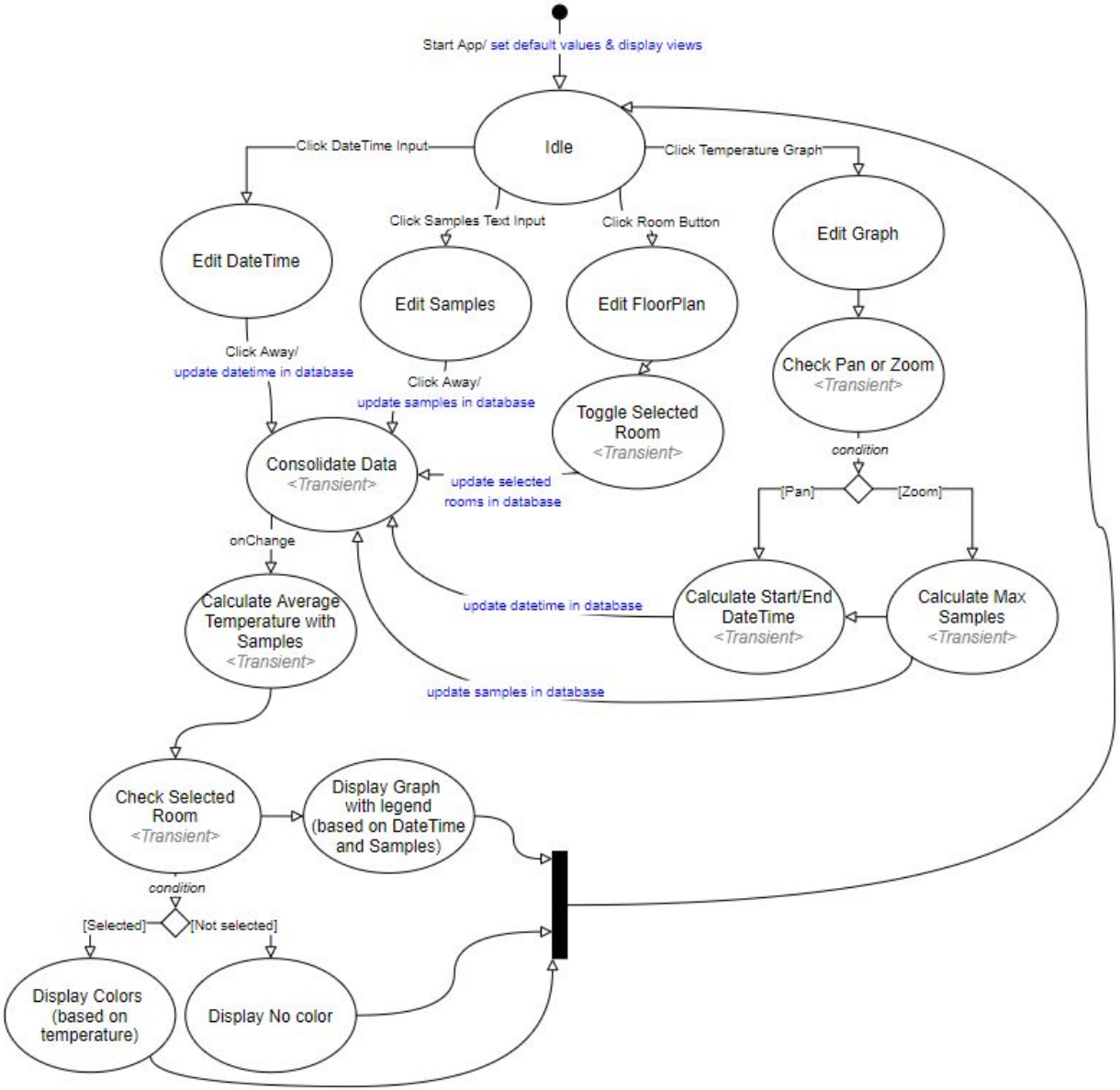
- Components have a local state used to update display view, where the data used is retrieved and updated from the server database
- Changes in TemperatureComponentView Model calls the updateStart() and updateEnd() to update the start & end datetime values in the server

# 4-Tier Architecture

https://app.diagrams.net?lightbox=1&highlight=0000ff&edit=_blank&layers=1&nav=1#G1uI9p1c9fHfEg6FRmU5kLrztBST-DSUHc

# Statechart

https://app.diagrams.net?lightbox=1&highlight=0000ff&edit=_blank&layers=1&nav=1#G1wLSOy-RSGsqXwbCquSZS0D-nzrZ27tDS



# Temperature Time Series Data schema

**Json Format:**

```
data: {
      _id: String,
      RoomId: integerString,
      timestamp: datetimeString,
      temperature: floatString,
}
```

e.g.

```
{
      "_id:" :  "fMryR3ycPEA5LDFE3",
      "RoomId": "6",
      "timestamp": "2013-10-02T05:00:00",
      "temperature": "20.615",
}
```

This format was chosen because Meteor uses MongoDB for data storage.

Thus, we are able to use their db.Collections.find({}) to sieve through data.

This can be done by filtering data that contain the "timestamp" field and retrieve only data that are greater than ($gte), but less than ($lte) and equal to a specific time range. It will also allow us to retrieve only the selected rooms through filtering the "RoomId" field.

# Implementation Details

## Define any Convenience Functions or Calculations

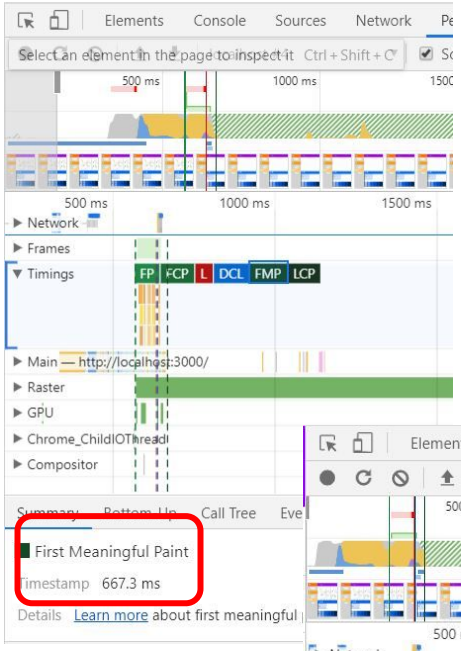
- client/main.js was used to render the view components
- client/main.html was used to order the view components and allow for nested flexbox (for **responsive design**)
- All .css files were used to align and clean up each view component
- room-temperature.csv is stored in a private directory to allow for the server to read the file and convert to json, to be stored in database on server startup in server/main.js
- Used ‘papaparse’ to clean csv file for storage and ‘dygraphs’ for time series graph with npm install

# Justifications for Architecture and Technology Choices

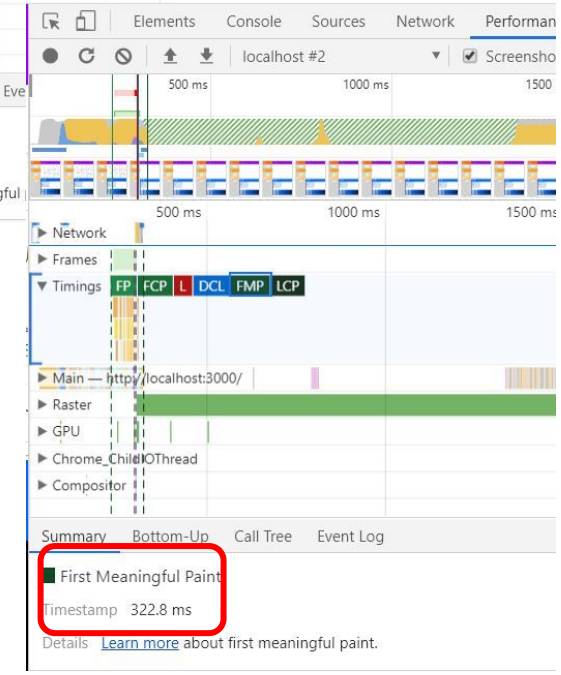
- MVVM was chosen because separate software components to be decoupled:
  - View components can be relatively light but have varied functionalities that are handled by the view model
  - Allow for smooth development of view components and their corresponding logic (in view model) separately
  - Unnecessary components are separated (e.g. business logic and data from ui)
- 4-tier architecture was used because it allows for separate hardware or software processes to be decoupled:
  - Client Tier can handle what the user will see
  - Web Tier can handle user inputs and function callbacks
  - Business/Logic Tier can handle the api calls and run the functions
  - Data Tier can handle the data at a consolidated location, where logic and data tier can be reused for multiple clients if needed
- Meteor was used because it allows front and backend development in a single language with full stack reactivity. It also supports data on wire allowing for more dynamic applications (and be **app-like**, with minimal refreshes)

# Performance Audit: Issue 1

## Reduce Downloads, Minify Text




**Before**




**After**

**Before:**

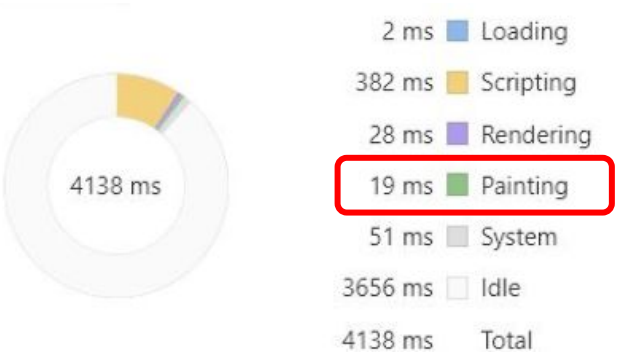
- Using Chrome DevTools, the initial implementation was seen to have a first meaningful paint (FMP) of **667.3ms**, which was due to the large main.css that was used to style the page. Although it was acceptable at <1s, we felt it could be faster and it could be done by reducing file download sizes.

**After:**

- Main.css was minified by shortening the string length through removal of whitespaces. This reduced the file size from 2.29KB to 906B. By reducing the file size download, we were then able half the load up time to hit **322.8ms** for the FMP allowing for faster load up.

# Performance Audit: Issue 2

## Skip parts of rendering pipeline, Update only changed paint




**Before**


**After**

**Before:**

- Using Chrome DevTools, the painting for the floorplan updates was seen to take **19ms**, which was due to the force repainting of the whole floor plan component when changes were detected, either from the server data or on click. Although the response was <100ms, we felt that it could be faster.

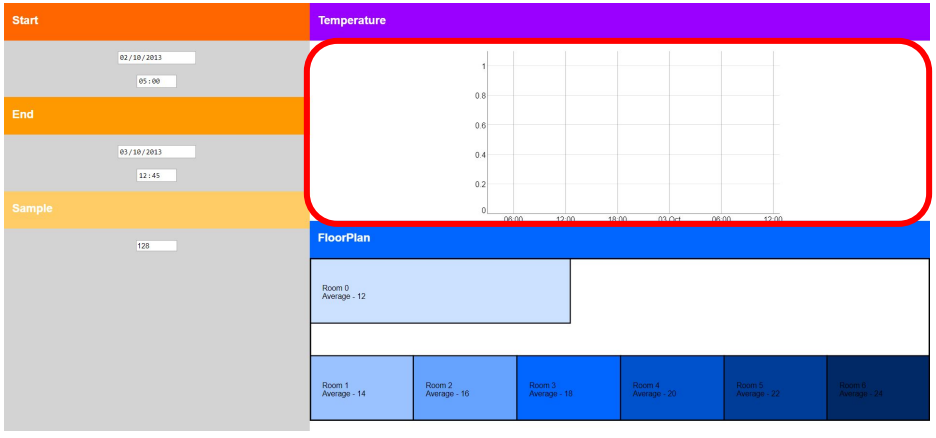
**After:**

- Part of the rendering pipeline was skipped where we only update paints that needs to be changed. This was done by changing the ‘color’ field of individual svg rectangle when the color is updated. This avoids needing to repaint the whole component and only update changed paint, allowing a faster repaint of **10ms**.

# Usability Review: Issue 1

**Issue:** Gulf of Evaluation, **Strategy:** Feedback

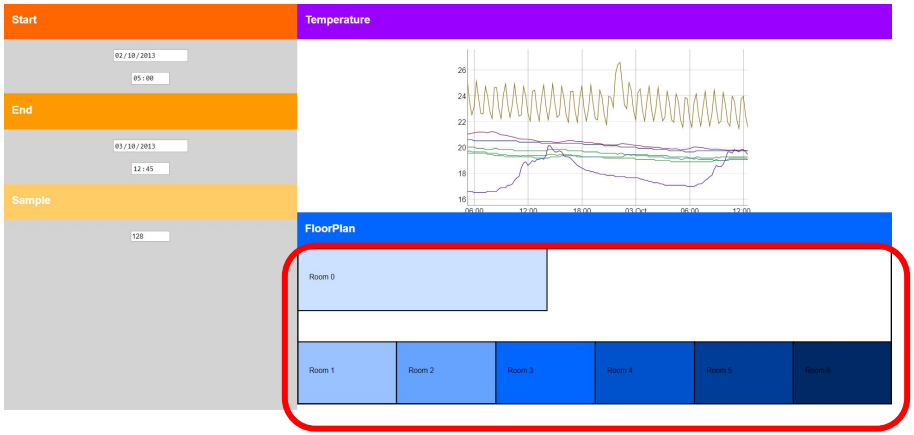
- When the application is started, the time-series graph at the temperature component takes some time to load. However, it does not give any feedback on the state during load up. This violates heuristic 1 of Jakob Nielsen's 10 Usability Heuristics as the lack of status feedback makes it hard for the user to know that the data is loading, or whether the component has frozen up, resulting in a large gulf of evaluation.
- A possible strategy would be to use feedback, such as an hourglass or even a progress bar, to let the user know that the component is working and that it is just loading up, so that they can get a better grasp of the current status.



# Usability Review: Issue 2

**Issue:** Mental Model Mismatch, **Strategy:** Skeuomorphism

- The application employs a very minimalistic and flat design. With rectangular boxes to represent each room and colour intensities to represent temperature, the design could be very confusing to new users and violate heuristic 2 of Jakob Nielsen's 10 Usability Heuristics. The reason for this is that the user might be expecting an actual detailed floor plan on how the rooms are situated, instead of displaying average temperatures.
- A possible strategy could be to employ skeuomorphism such that the room outlines follow a design closer to an actual floor plan, while each room could have a temperature logo that indicates the average temperature of the room instead of solely relying on color.



# Teamwork Contributions

| Adil Bin Azmoon | Lim Yong Song | Combined (Both) |
|---|---|---|
| Skeleton Draft *(Report)* | Implementation Details *(Report)* | UI Design Improvement *(Report)* |
| Data Schema *(Report)* | Justification *(Report)* | View Tree *(Report)* |
| Data Retrieval *(Code)* | Skeleton Files *(Code)* | UML Class Diagram *(Report)* |
| Temperature Component *(Code)* | Data Storage *(Code)* | N-Tier Architecture *(Report)* |
| TimeSeriesGraph (*Code)* | Start/End/Samples Component *(Code)* | Statechart *(Report)* |
| | FloorPlan Component *(Code)* | Performance Audit *(Report)* |
| | | Usability Review *(Report)* |
| | | Lessons Learnt and Takeaway *(Report)* |
| | | Business/ Logic functions *(Code)* |
| | | Cleanup & Bug Fix *(Report/Code)* |

# Lessons Learned and Module Key Takeaway

- Anyone can be a good designer, and not just people with artistic talent, as long as we follow the design principles (e.g. Principles of Composition)
- There are alot of frameworks and widget toolkits (e.g. Flexbox, React) that can help simplify ui development
- Following design patterns like MV* can help create better functioning ui that can be easy to develop and maintain
- Other than decoupling software, we can also decouple hardware by utilising N-tier architectures which can help create better applications (e.g. faster, less data-usage) depending on user needs
- There are also many types of web stacks (e.g. Meteor) that we can use, and each have their own advantages
- We also learnt that performance issues can have a huge impact on usability (with e.g. Chrome DevTools) and there are different areas and ways which we can alleviate this
- On top of all these, it is important to design for the user by using heuristics (e.g. Jakob Nielsen's 10 Usability Heuristics) and user tests to make sure we do not fall into the false-consensus effect

# README: project files organization

- adilbinazmoon_A0155858M_limyongsong_A0154776R_Assignment4/
  - <*misc: .meteor/; node_modules/; test/; package.json;...>
  - client/
    - main.css; main.html; main.js
  - imports/
    - api/
      - tasks.js
    - ui/
      - endcomponent.css; EndComponent.js; floorplancomponent.css; FloorPlanComponent.js; LeftPane.js; RightPane.js; samplecomponent.css; SampleComponent.js; startcomponent.css; StartComponent.js; Task.js; temperaturecomponent.css; TemperatureComponent.js
  - private/
    - room-temperatures.csv
  - server/
    - main.js

Notes:

- Files in 'client/' help to organise and display each component at site startup
- Files in 'imports/api/' store the functions that are should be run on the server and are mostly used to update the database
- Files in 'imports/ui/' are the view and viewmodel components that make up the site, each file does function callbacks to update the database when changes are made by user
- Files in 'private/' holds the csv files that needs to be stored into database
- Files in 'server/' initialises the database at server startup