Design Doc for SpringMVC

Run the code:

```
# At root of project
gradle bootRun
```

Login

After running `gradle bootRun`, Spring start the app. In your chosen browser, go to `localhost:8080` to login to the app. The preset Admin(Marker) account will be

Username: admin Password: seng330

Acceptance Tests Focused On

- Scenario A: 1,3
- Scenario B: 2,3,4,6
- Scenario C: 1
- Scenario D: 1
- Scenario E: 1,2
- Scenario F: 1,2,3

Unit Tests

At the root of the code run `gradle test`

Why Spring?

Our chosen framework is SpringMVC. Due to the experience of the developers on the team, a web-based framework would be easier to incorporate into the project out of the two options. Furthermore, when planning the initial design of the project, current developers of companies in Victoria were asked their opinions on SpringMVC and JavaFX. Every person asked stated they've heard good things about Spring from coworkers who have used the framework, and they hadn't heard about JavaFX.

Dependency Injection

Currently, the app will hold information on the system whenever it transfers between pages. With the use of `dependency injections` we could ensure that `hub` will only need to be initialized once and then used consistently throughout the system until the page is reloaded. This is done using the `HubConfig` file, which uses `@Configuration` to create a `@Bean` of hub.

Template Pattern

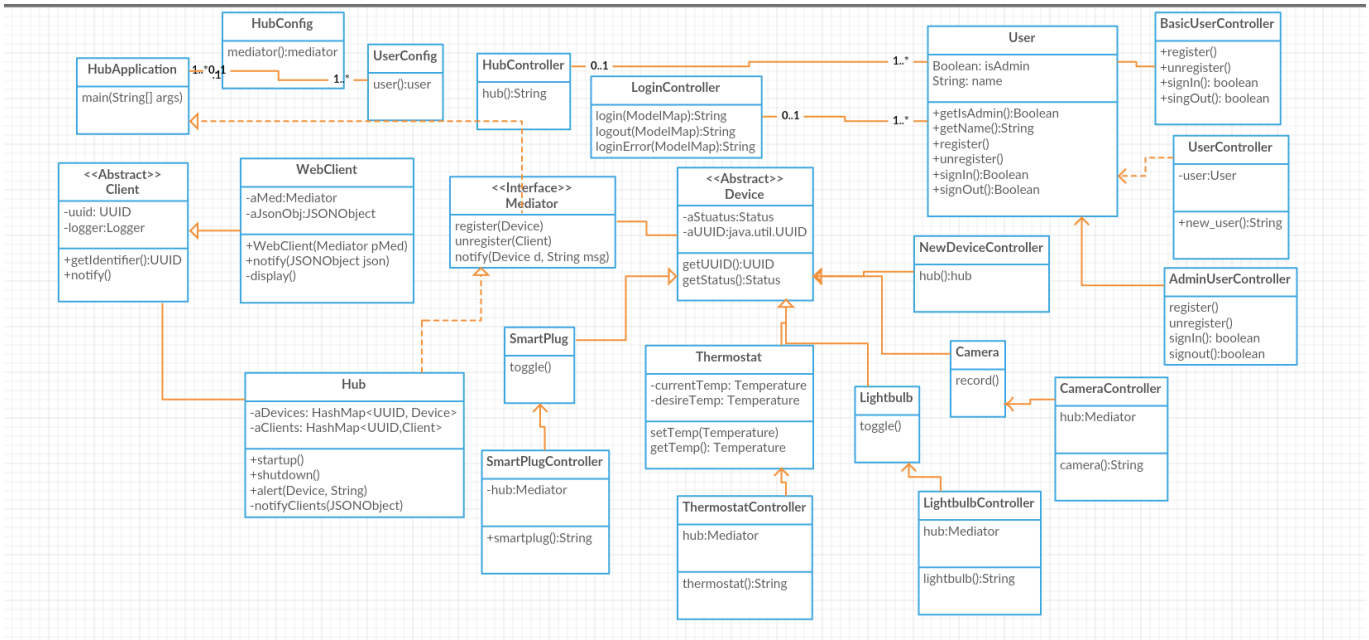
Coming from a co-op where one project used KnockoutJS (KOJS) with `html` templating, the templating pattern seemed familiar. KOJS uses `data-binding` in order to talk between parts of the system. The controllers used to speak between the models and views (`html` templates) have the same functionality. Due to this direct communication, the user doesn't need to be aware of how the system works.

Model

Consists of the retrieval and storage of data that our application uses.

Controller

Consists of handling the user input data and the logic to complete each task the user wishes.

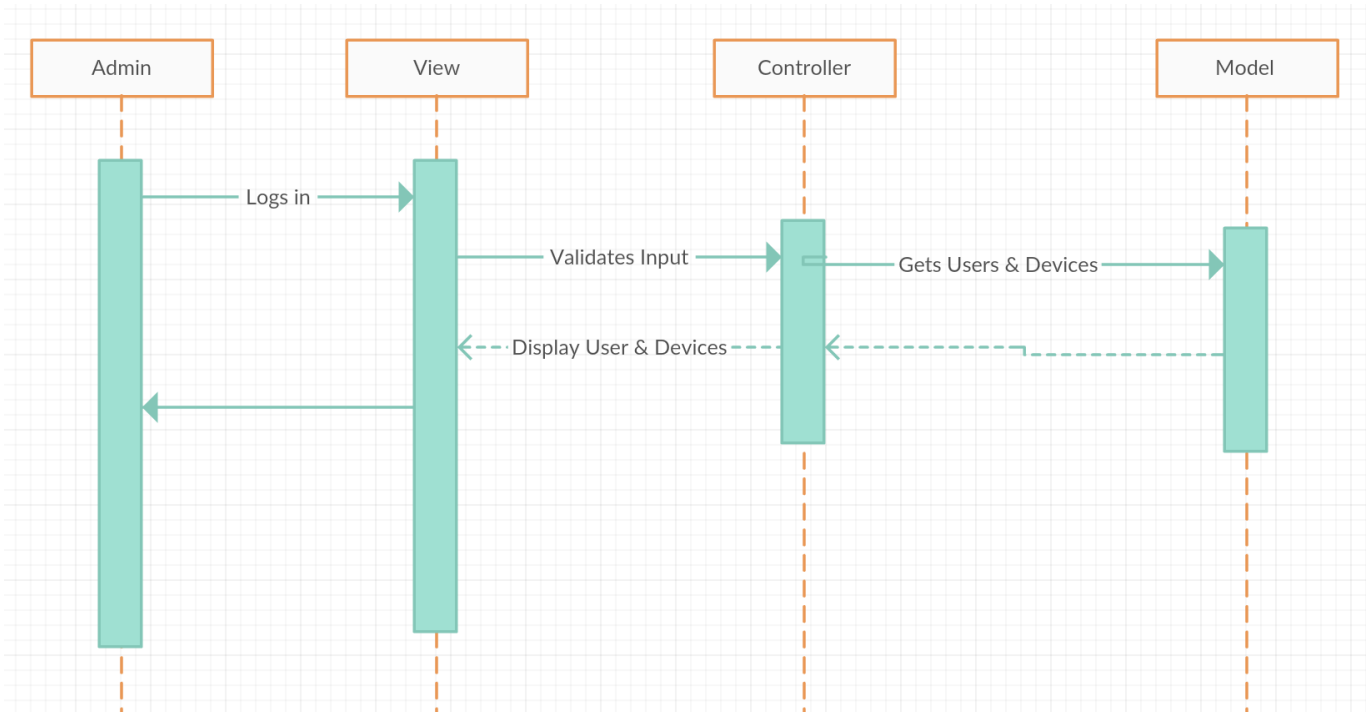


View

Presents options for the user to select or forms for the user to complete. It then presents the results of the users choices back to them.

Walk-Through Scenario A: #5

- View presents Log-in page
- Admin logs in
- View Validates this data and triggers the controller
- Controller requests users and their device information from model
- Model passes this data back to the controller
- Controller passes this back to a new view
- View presents this data to the Admin


Admin
View
Controller
Model
Logs in
Validates Input
Gets Users & Devices
Display User & Devices