Initial software implementation & testing：

1. Architecture / design / proof of concept implementation

Architecture:

Design:

Our team has preliminarily designed the interface

Sponsor changed the design and provide example structure

Present implementation and future strategy (Diff components)

Proof of concept implementation

Original user requirement and implementation

Present user requirement and implementation

1. Data storage / management / manipulation / use of (personal) data (if applicable)

We made out the data

We use php files to set up database automatically

1. Appropriate use of version control system (branching / merging)

We divided the project into three stages

Developed six parts components individually

After the first part, use dev to build combination code

1. Evidence of testing

JS file for calendar testing

PHP file for personal profile testing

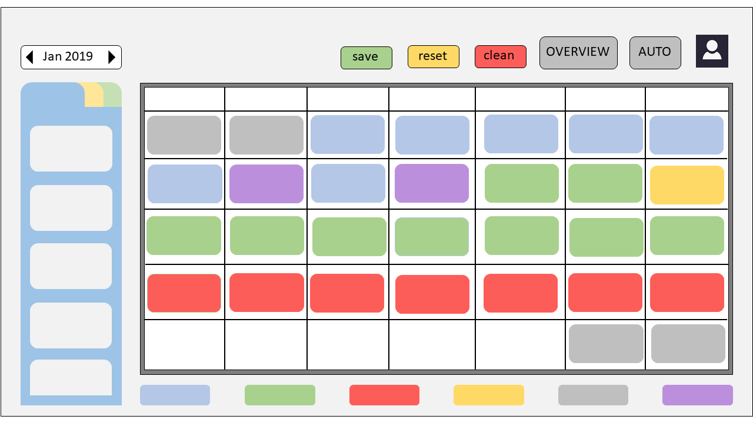
JS file for log in system testing

1. Preliminary inline software documentation

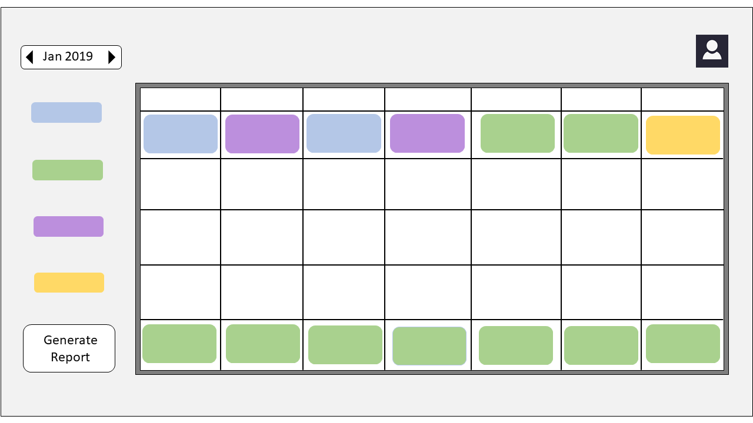
Only when the logic is confusing

Initial software implementation & testing：

The design of architecture of out project started from the very beginning presentation. After great amount of investigation amongst present commercial rostering system which we could find online, we have generated a rough designed prototype of this system. Comparing the information from the project description with the example products we investigated, we inferred that the most significant property our sponsor longing, is to save his or her workload on manually checking several requirements of employees to set up schedules. Below is our original design of the administrator interface:



The employee interface:



(Further details?)

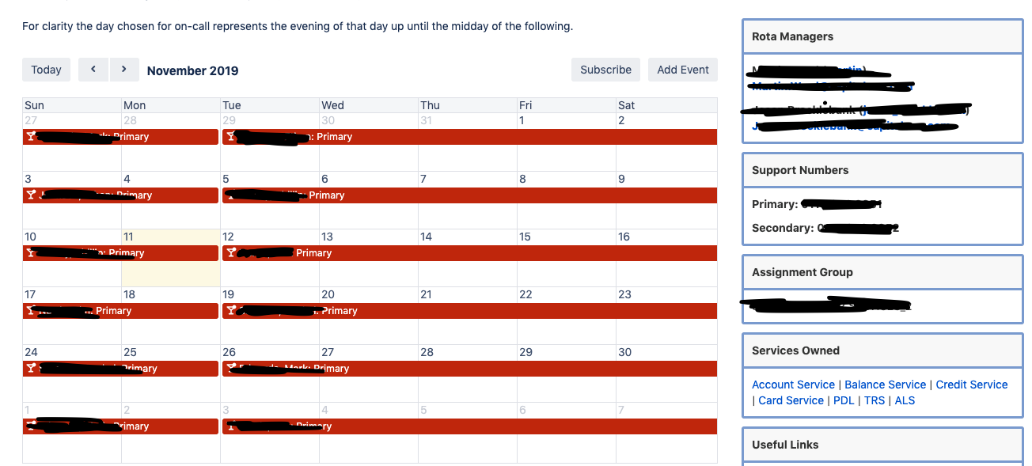
According to the “Call for Expressions of Interest” of our project, the employee property needs to be evaluated is listed:

1. Whether the employee is in probation period (3 months after join in);
2. Whether the employee’s status is above ‘strong’, the status changes automatically;
3. Whether the employee is on holiday or work immediately after it;
4. Whether there exists deployment within their scheduled period;
5. Same employee amongst n could not be allocated jo within n-2 weeks;

Present user requirement:

1. Only the administrator could manipulate the status of the employee. And the status could be “Active” & “Inactive” which substitutes the “Strong” & “Below strong” status and probation period evaluation;
2. The holiday and deployment periods are still important and should be kept individually;
3. All employee should be able to view this timetable without log in;
4. The administrator should log in to change the status of employee;
5. The administrator should log in to generate report of single employee;
6. The administrator should log in to change the schedule.

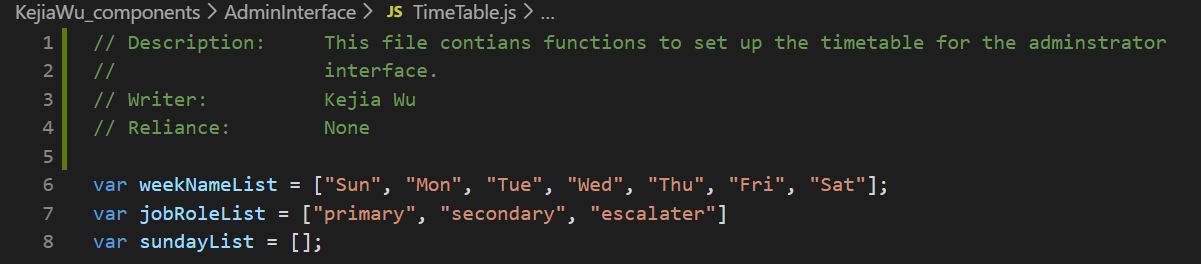
Administrator interface:



Present workout:

Preliminary inline software documentation

According to clean code, the inline documentation should be as less as possible. We decide to write most part of the commit in the begin of files to demonstrate files’ functions, reliance and other useful information (e.g. the idea of better implementation).



The inline documentation will describe some complex computation or logic implementation. They were short, specific and rare to see within the entire project since it is a web application project which should not contain lots of calculation work.m

