ShiftShark Teamwork Plan

6.170 Software Studio | Project 4.1

Authored by: Andre Aboulian, Cathleen Gendron, Elliott Marquez, Michael Belland

---

Stakeholders

Employers/ Project Managers**:** Employers are our audience. They want ShiftShark to be a way to quickly and easily manage job scheduling for their employees, allowing them to focus their management work away from the ever-recurring scheduling problem to where their focus is most needed.

Employees**:** Employees aren't really here by choice -- their employer signs them up for the system, after all -- but the system relies on employee interaction. If employees don't report their trades on ShiftShark, requiring employers to update the schedules, employers will likely find the system to be at least as hasslesome as other scheduling solutions.

Resources

ShiftShark doesn't require any external API, so costs are limited to internal development costs (which every team has). Our proof-of-concept/MVP uses a data model implementation intuitive to users but not to programmers (because simpler models seem to exist at first glance, although we believe that our model provides more user freedom and flexibility), so the cost to develop may be higher than average. Time constraints should be no different from those of other teams, but we anticipate that the deadline for the MVP, especially given the fact that we'll use it to test a difficult data model, will be particularly tight.

Computational constraints are mostly limited by the database queries we would need to make to populate all the shifts for a given day or when a shift is divided for trading purposes. Our model optimizes for space -- rather than having potentially 168 different shifts per instance of a role (one for each hour for each day in a week), our model allows for flexible shift times. By using linked lists and mongo queries we anticipate that looking up shifts, even when split due to trading, should not be computationally intensive.

Tasks

Backend

1. Write Mongoose schemas to reflect our data model. *2 hours, Cathleen*
2. Write specifications for API routes. *8-10 hours, Cathleen & Andre*
3. Implement user authentication with PassportJS. *1 hour, Elliott*
4. Implement ability for privileged user to create employees and invite them via email. *4 hours, Michael & Elliott*
5. Implement ability for privileged user to give privileges to other users. *2 hours, Michael*
6. Implement employee's ability to declare availability. *6 hours, Andre & Elliott*
7. Implement ability to assign shifts to employees. *6 hours, Cathleen & Michael*
8. Implement shift offering. *2 hours, Andre*
9. Implement shift claiming. *2 hours, Michael*

Frontend + UI

1. Write template for displaying the weekly schedule. *4 hours, Cathleen & Andre*
2. Write template for displaying a single employee's shifts. *2 hours, Elliott*
3. Implement UI for employee availability declaration. *4 hours, Michael & Andre*
4. Implement UI for offering shifts. *4 hours, Elliott & Cathleen*
5. Implement UI for claiming shifts. *4 hours, Elliott & Cathleen*
6. Implement UI for administrative dashboard - assigning shifts to employees. *4 hours, Michael*
7. Implement UI for administrative dashboard - creating new employee accounts. *2 hours, Andre*

Milestones:
11/14 - backend tasks completed
11/17 - frontend tasks completed
11/18 - testing & bug fixing; MVP due at 11:59PM

Tasks after MVP (subject to change):

11/19 - Meet as team to discuss recommended Data Model/Design changes
11/21 - Write tests for major back-end function calls
11/23 - Compose security tests
11/25 - Revised Design Doc due at 11:59PM (Project 4.3)
11/28 - Implement Availability, Security Tests
11/29 - Determine if allowing employers to automatically create shifts w/availability feasible
11/30 - Implement Automatic Shift Assignment (should allow peer-to-peer shift planning)
12/2 - Code for Grading complete due at 11:59PM (Project 4.4)
12/5 - Improvements to UI, and final feature adding deadline
12/7 - Production code complete, Practice presentation for Project Fair
12/8 - Project Fair

Risks

Shift Implementation

Our intent in modeling shifts was to support the type of interactions a user would expect with recurring events in a modern calendar application. As we discussed at length in our design challenges, we faced difficulty in determining a structure that would allow for deviations in recurring shifts. This is crucial functionality, since traded days and deleted days (holidays, for instance) constitute deviations from the regularly recurring shift. We intend to implement a recurrence as a linked series of shifts pertaining to that recurrence. Although we have determined this to be the ideal representation, it is inherently complicated to implement, especially with regard to handling pointers between shifts.

We have decided to consciously limit the time we spend attempting to implement this approach, reverting to a much simpler model of recurrence once we realize there are too many edge cases. This simpler model would entail creating individual Shift instances for each day. This would require the employer to select fixed start/end dates over which shifts should be created, but recurrence would only occur in the sense that there would be copies of the shift for every week. This would not support infinitely recurring shifts, and would prohibit users from editing multiple weeks of a shift in a single action. However, deviations become trivial to implement, since an individual shift (on a single day) could be deleted/traded/shortened without affecting any others.

Intuitive UI Design & Implementation

One of the core aspects of ShiftShark is ease of use for both employers and employees. In order to achieve the desired level of simplicity, we hope to implement a user interface that is focused on intuitive and visual drag-and-drop elements. For example, if an employer wanted to assign a shift to an employee, they could click and drag to "paint" the timeslots for a given shift, or drag-and-drop entire shifts among employee's slots.

Implementing these UI elements will likely be tricky, and will rely on external resources such as jQuery UI. Introducing this extra layer of complexity could be very time-consuming, and could potentially cause a lot bugs. If we find that implementing this more intuitive kind of UI is becoming too time-consuming, or is causing more trouble than it is worth, we will default to our simpler, MVP user interface, in which a user enters data through usual means, such as drop-down menu values or text entry fields. These simpler methods of data entry will be more straightforward to implement, and will be less time-consuming for our team to complete.

Minimum Viable Product (MVP)

Our MVP stands to serve as a proof-of-concept for ShiftShark, and thus must go to show the viability of ShiftShark in satisfying the first two of its stated purposes. Thus, we will focus on limited actualization of the “Shift,” “Trade,” and “Schedule” concepts. In our MVP, times of shifts will not be able to be split for the purposes of trading, and thus only whole shifts may be offered or claimed. To be more explicit, if I have a 9am-12noon shift for all four Mondays of a month, I can trade my shift on the second Monday, but I can’t trade a 10am-11am portion of one of my Mondays. This will give us the chance to experiment with the most challenging issue we face -- dividing a shift into three shifts -- without dealing with the more complicated issue of dividing a shift into three non-recurring shifts for the purposes of trading.

We will postpone security issues. The MVP will focus on developing a useable UI, postponing a stylish and intuitive UI to later phases of the project. The schedule should be able to display shifts, and clicking on a shift should allow the user to interact with it, but complex user actions such as dragging-and-dropping to create new shifts will not be implemented for the MVP. Necessary actions will, at worst, be able to be performed by interacting with buttons on the UI.

Because the MVP will allow users to trade shifts amongst each other, even with limited capability, and will have a user interface (no matter how rudimentary), we expect that it would be useful. Of course, by polishing the user experience, enhancing our design by adding the concept of availability, and possibly adding other features (collaboration across multiple schedules comes to the team’s mind), we believe that our final product will build significantly upon our MVP and provide even more value to our users. By designing our fundamental concepts for ShiftShark directly into the MVP, we believe that both our code and our user experience can receive meaningful feedback that can help the MVP evolve into the final product.