***ShiftShark* Design Document**

6.170 Software Studio | Project 4.1

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

---

Motivation

ShiftShark is a web application designed to manage work schedules. It provides a platform for employers to construct schedules, allows employees to adapt working hours as needed, and communicates a single unified schedule among employers and employees. ShiftShark is oriented toward small workplaces, where employees do not perform fixed roles.

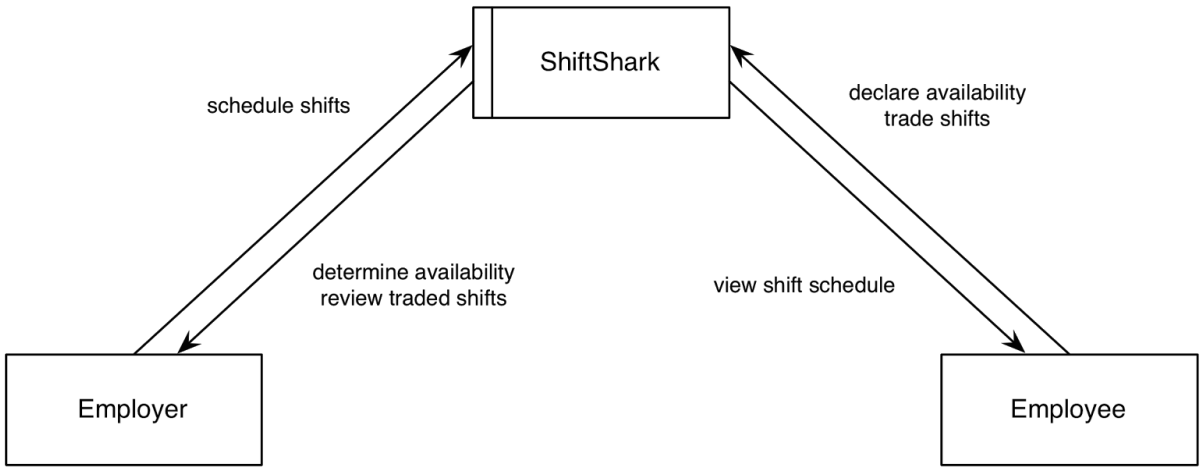
The purposes of ShiftShark are as follows:

1. **Allow employers to assign work hours for employees.** ShiftShark acts as a platform where employers can easily assign and change their employees' hours, in a centralized location visible to everyone. At a quick glance, an employer will be able to see whether they have the workers they need when they need them, and employees will be able to see when they needs to be at work.

2. **Enable employees to exchange working hours amongst each other.** Employees will be permitted to give up hours they have been assigned but are unable to attend. Coworkers will have the opportunity to cover these hours on their behalf. This interaction occurs transparently, but without requiring intervention from the employer. It will no longer be necessary to email colleagues or call your boss to exchange working hours.

3. **(Potentially) Enable employees to communicate when they are willing to work.** Employees will declare which days and times they are able to work. Their employer can proceed to schedule hours with this data in mind. By integrating this functionality into the application, we prevent employers from needing to survey their employees externally and reduce the likelihood of hours needing to be exchanged.

Context Diagram



Concepts

Shift: A single unit of work that can be scheduled, and subsequently traded. A single shift denotes the time range on a specific date that an employee is scheduled to work, and what "position" they are working in. Shifts can only be assigned by a user with special privileges (usually the employer). This construct allows employers to easily assign work hours to employees.

Trade: The action of giving a shift to another worker. Trading is composed of two steps: first, one user "offers" their shift to the other workers; then, another worker who sees the shift and is willing to take responsibility for it "claims" the shift. Notably, a trade is not a swap, so the worker who claims a shift does not need to offer any shift to complete the trade.

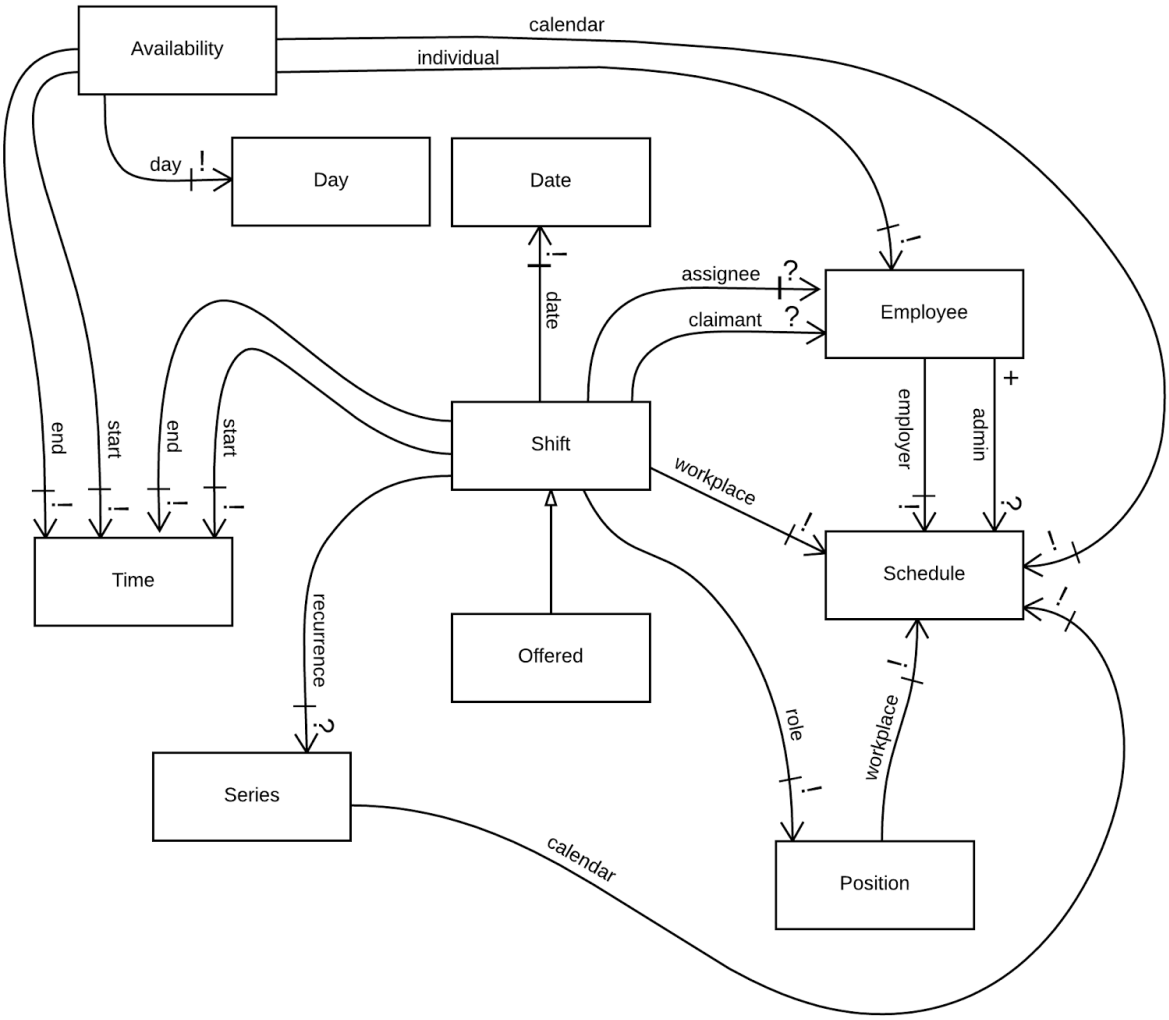
Series: A collection of shifts with that correspond to one particular type of job that happens over a given period of time. The job must be repeated on the same day of the week for multiple weeks, and each of the shifts in a series must have the same start and end times. However, the weeks for the shifts in a series do not have to be consecutive.

Schedule: The entity that contains a collection of shifts, a schedule unifies employees into one entity. We envision a schedule grouping together an employer and employees with each other and their roles, but it could also be used to manage tasks and roles for subdivisions of a large company. By providing the one unifying factor bringing all workers together and by gathering weekly employee shifts into a coherent calendar, this construct is necessary for assigning work to employees.

Availability: The times that a worker states they are available to work. These times are laid in a weekly schedule, where an employee can state what times they are available on each day of the week. This weekly availability schedule represents when an employee is available on each day for any given week.

Position: A role to be performed for the workplace. A position consists of one action that an employee could perform for his work (e.g. cashier or IT phone call assistant). Positions provide logical organization and meaning for each shift.

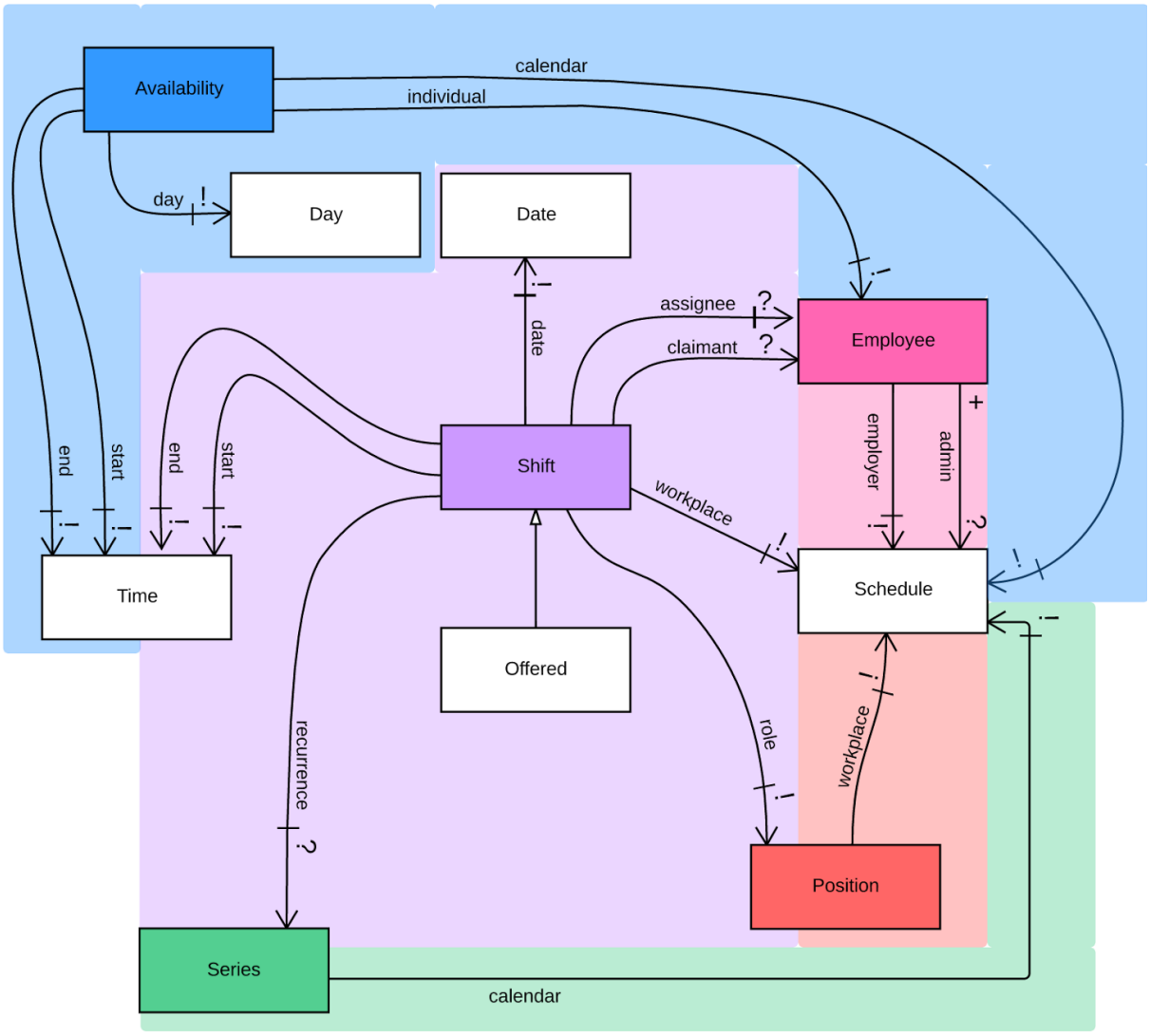
Data Model



Additional Constraints

- Starting time/date for a Shift or Availability must occur before the ending time/date.
- Each Series must reference the same Schedule as its corresponding Shifts.

Data Design



Security:

Key Security Requirements

- Only administrators can create, edit and assign shifts.
    - We address this by creating an access control list of people that are administrators. We can call this ACL by searching our database by schedule name, and filtering by a flag on each user that indicates if the user is an administrator. Additionally, each administrator will see an administrator view of the page which has the options that are not available to non-administrators.
- Administrators must be able to control the admin ACL

    - We address this by only allowing current administrators to add and remove other users added to the administrator ACL. Every schedule must have at least one administrator and is created with at least one administrator, so someone always has control over the administrator ACL.

Standard Attack Mitigation

- Injection Attack
    - Only inputs that would be vulnerable would be name, schedule name, email, and password entry. Passwords would not be an issue because we are using passport which will hash the password and thus turn it into something that is not a threat. Name entry and schedule name entry will be sanitized before inserted or displayed on a page. Email will also be checked for email structure on the client side and then again on the server side and then finally sanitized before it is inserted anywhere.
- Cross Site Scripting
    - The best way we can mitigate this would be by sanitizing the inputs listed in the injection attack. This is because XSS would be most common in data insertion which is then later displayed. We can also have a check that will warn users when they are being redirected outside our site. Additionally, we will use a private API that uses hashes and non-identifying requests when appropriate.
- Cross Site Request Forgery
    - We can mitigate this by sanitizing the inputs listed in the injection attack. We will also check that every request made is authenticated using the passport session cookies which are encrypted. Additionally, we will use a private API that uses hashes and non-identifying requests when appropriate.

Threat Model

- Potential Risks
    - XSS, CSRF, Injections
    - Admin abusing admin powers, or admin account accessed and abuses power.
    - Someone gains access to a normal user account and abuses the account's power (e.g. trades all shifts / claims all shifts)
- Assumptions
    - No sensitive data stored on our servers other than names, no need to assume we are a big target for hacking attempts.
    - People in corporate / job world hate complex password requirements
    - Disgruntled employees may abuse administration powers
    - Administration has a mental idea of how the schedule should look
- Mitigations
    - Standard cross site sanitization and mitigation techniques should suffice.
    - Have an email password change / reset. Also email out all account changes and attribute the changes to a specific user. This will allow users to be notified

if someone is changing the admin ACL and reduce the risk that one person will have all of the administrative powers to themselves. This makes it easier for the issue to be resolved in person.

- Have an email confirmation sent to a user and administrators when an account claims or trades a shift. Receiving an inordinate amount of these emails would indicate that something out of the ordinary is happening. Additionally, an administrator can edit shifts however they want so they can undo most of the changes made maliciously, and they have an email log of the changes made. Finally, administrators can delete accounts and the employee can change their password.

Design Challenges

How should shifts be modeled to support recurrence?

Issue

Our most difficult consideration involved the concept of a Shift, which is central to the application. Shifts in a workplace are often scheduled on a recurring basis, although deviations from this pattern may occur during a specific week. Our goal is to represent shifts in a way that does not inhibit employers from scheduling shifts that occur once, recur, or differ from a recurring pattern.

Option: Master Calendar

Our first approach involved a *Master Calendar*, in which the employer would enter (indefinitely) recurring shifts. Temporary deviations -- not working a certain week or trading a particular shift -- from this schedule would be stored separately. To obtain an accurate schedule, the deviations must be superimposed onto the master calendar. Of course, one-time shifts would be modeled as individual deviations.

Although this strategy would satisfy our intended modes of shift scheduling, it inherently suffers from a versioning problem. That is, the deviations are only relevant to the *current state* of the master calendar. For instance, if a shift for a new employee is added, it would appear in all future and previous weeks, which is inaccurate. A proposed solution to this could entail copying the schedule each week before making changes. This would become unmanageable complicated, since changes made to the master schedule would also need to be copied to each future copy. Future deviations may also become invalid when shifts are removed from the master calendar. Additionally, recurring shifts cannot begin in the future, since the master schedule is active beginning in the current week.

Option: Master Shifts

An alternative approach would be to have shifts that could start on any arbitrary date; they would recur indefinitely or have a fixed end date. Individual deviations from this recurrence would need to be stored separately. These deviations would

be unified under the *parent* shift they affect, which can be thought of as a *Master Shift*. This would obviate the versioning concern expressed earlier, since the master shift has defined start/end boundaries during which a shift recurs. And all deviations can be traced to this master shift, which "contains" them.

Almost all possible deviations are easily implemented in this model. A traded shift would simply be a non-recurring shift with the recurring "master" shift as its parent. Truncating the start/end dates would simply involve changing the limits on the master shift and removing all deviations that fall out of these bounds. The only inherent problem with this strategy would be adjusting the time of a shift for all future weeks. This would require an infinite number of deviations to the database. This specific problem may be mitigated by splitting the shift at the point this change occurs, forming two separate shifts -- one at the original time and one at the new time. However, there would now be no unifying factor between the two "halves".

Another issue with this model is that querying the shifts does not express the full picture, in that deviations (especially deletions) must be superimposed onto their master shifts. This means that partial data may be mistaken as complete.

Option: Linked Shifts
Another approach entails creating shifts with a defined start date and a fixed/infinite end date, as before. A series of these shifts can then be formed similar to how a linked list operates.

If a particular day of a shift is modified, that shift is *split* (into **A** and **A'**) excluding the desired day, and a new shift **B** is created for the specific day. Then, much like a linked list, shift **A**'s "next" field points to shift **B**, whose "next" field in turn points to **A'**. In effect, the unique shift is "wedged" into the pattern. In the case of a deletion, **A** would be directly linked to **A'**, effectively skipping a week.

Notice that this strategy does not require any form of deviations; the recurrence is expressed more literally by using a series of linked shifts. To find the end date on a given recurring shift, the links simply need to be followed to the end.

(Chosen) Option: Individual Shifts & Series

Each of the previous approaches emphasizes space efficiency; by doing so, they are complex and difficult to understand/implement. Alternatively, we can define our Shift to be an individual block of time, occurring once, on a specific day. To form a recurrence, we simply create a new Shift for the same day/time each week within the recurring period. Since we treat each instance individually, this strategy makes it simple to delete a single shift, modify it, or trade it without affecting any around it.

However, this approach reintroduces the problem of being able to relate shifts within a recurrence; the individual Shifts cannot be correlated by any means

other than crude filtering and guesswork. For this, we introduce the concept of a Series. Each Shift within a recurrence will point to the same Series, whose sole purpose is to enforce a one-to-many relation between itself and specific shifts. Filtering on the Series will yield all Shifts within that recurrence. This is crucial for knowing when the recurrence starts/ends or for deleting/trading a portion of it.

What constitutes employers and employees?

Issue
Given the application's context, employers and employees interact with the application in different ways. Their representation in the system should reflect their distinct roles. Additionally, both supervisors and companies are thought of as "employers", and this definition should be reconciled.

Option: Separate Employer & Employee Accounts
Our initial approach was to utilize separate user (employee) and admin (employee) account types. This strict dichotomy prevents employers from being employees themselves. It also forces the Employee accounts to work *for* their Employer account, meaning that the Employer account is the closest representation of their workplace in the model. It becomes unnatural to attribute Positions, Shifts, and Employees to an administrative account (Employer).

Option: Employers as Subset of Employees
By forcing a one-to-one relation between User and Employee, we ensure that each user on the system has the ability to work. The employer then becomes a subset of employees, correcting the issue of employers being unable to schedule shifts for themselves. However, this doesn't eliminate the construct of Employees working directly for an Employer account.

(Chosen) Option: Schedule Entity & Employer Permissions
In reality, employees are often working for a company rather than an individual employer. The idea of a workplace is captured by the addition of the *Schedule* entity. Employees (one-to-one for each User), Shifts, and Positions would be unified under a particular schedule, independent of the account that is administering the schedule. This changes the act of being an employer into an issue of administrative privileges; any user account could "own" (and thus, administer) a schedule and its associated resources.

Although not within the planned scope of the project, the separate *Schedule* entity even allows for the collaboration of multiple administrators on a schedule. It also allows an employer to work in more than one schedule (multiple workplaces).

How should trading shifts be modeled?

Issue

Aside from the problem of modeling shifts, one must consider how to represent trading on a particular day of a shift. Exchanging shifts between employees involves a component of social responsibility, which must be upheld by the system.

Option: Replacing Employee

Our initial approach was to simply replace the employee on a shift once it had been successfully traded (offered by one employee and claimed by another). This would be effective in tracking who is responsible for covering the shift, and each successive trade would entail replacing the employee again. However, we lose information when replacing the original employee. An employer would no longer be able to see the shifts as they were assigned or whether a particular employee trades shifts excessively.

Option: Backlog of Blame

Another approach is to preserve the original *assignee* but maintain a *backlog* (list) of employees who claimed the shift. In this version, the claimant would no longer be responsible for the shift if they chose to re-trade it. Instead, they would be logged and the shift would appear as if the assignee was still offering the shift. Socially, re-trading (especially at the last minute) would return blame to the assignee. Although the employee who gave up the shift after claiming it would be visible to the employer, the shift would likely go uncovered in reality.

(Chosen) Option: Assignee & Claimant

We believe that the ideal approach is a hybrid of the previous two options: we would like to preserve the original *assignee* while attributing blame to the latest *claimant*. In other words, if a claimant decides to re-trade a shift, they are responsible for finding a substitute. This substitute would, in turn, become the claimant and be responsible for attending the shift or trading it an additional time. The chain of blame is no longer relevant (or socially responsible), because the employer will look to the claimant. Although the assignee relinquishes responsibility once someone claims the shift, remembering who was assigned to work would be relevant to an employer.

Should the user be able to change the time of a shift?

Issue

Inevitably, an employer will need to modify a schedule to meet the company's needs after unexpected occurrences (such as client deadlines or inclement weather) occur. How quickly an employer will be able to adjust their schedule depends on the amount of freedom they are given in modifying shifts, but the underlying structures and assumptions of the data model must be preserved and not exposed.

Option: Time should be mutable

We initially thought of going this route. If an employer wants to push back the start of a working day by an hour or shorten an employee's shift so the employee doesn't work over 40 hours in a given week, it makes sense to give the employer the power to do so. However, we realized a fundamental conflict with this approach and our "series" concept after discussions during our TA meeting. For instance, if Jane's shift is moved to be an hour earlier than usual, that shift no longer would be a part of the series it belonged to (as the start and end times are different); but if an employer deletes the series, they probably intended to delete this modified shift of Jane's. Since we abstract away the concept of a series the employer won't know this, which will inevitably create problems.

(Chosen) Option: Time is immutable

Although this option is less convenient for employers, it ensures that their decision to modify a shift is a conscious one. Individual changes to a shift can be made by deleting an old shift and adding a new one with updated information; similarly a whole series can be changed by deleting it and creating a new series with the desired new time -- overall, this requires only a few more actions for a large return (the employer is aware of shifts that series actions won't modify).

How should we represent time of a day in our backend?

Issue

Our product interacts a substantial amount with time and varying intervals of a day. Because of this, we need a way to represent time in a way that we can easily manipulate, index, and display accurately.

Option: Use a Date object and parse the time of day

This was the initially more obvious route to take since the Javascript date object is very modular and has options to pull out time and seconds from the given object. This allows us to manipulate the time easily, and make sure we have a perfectly accurate representation of the time, but it is difficult to index ranges of time given our choice of shift modeling. Additionally, this approach would couple minutes with the date which would cause a lot of implementation issues when it comes to repeating a shift.

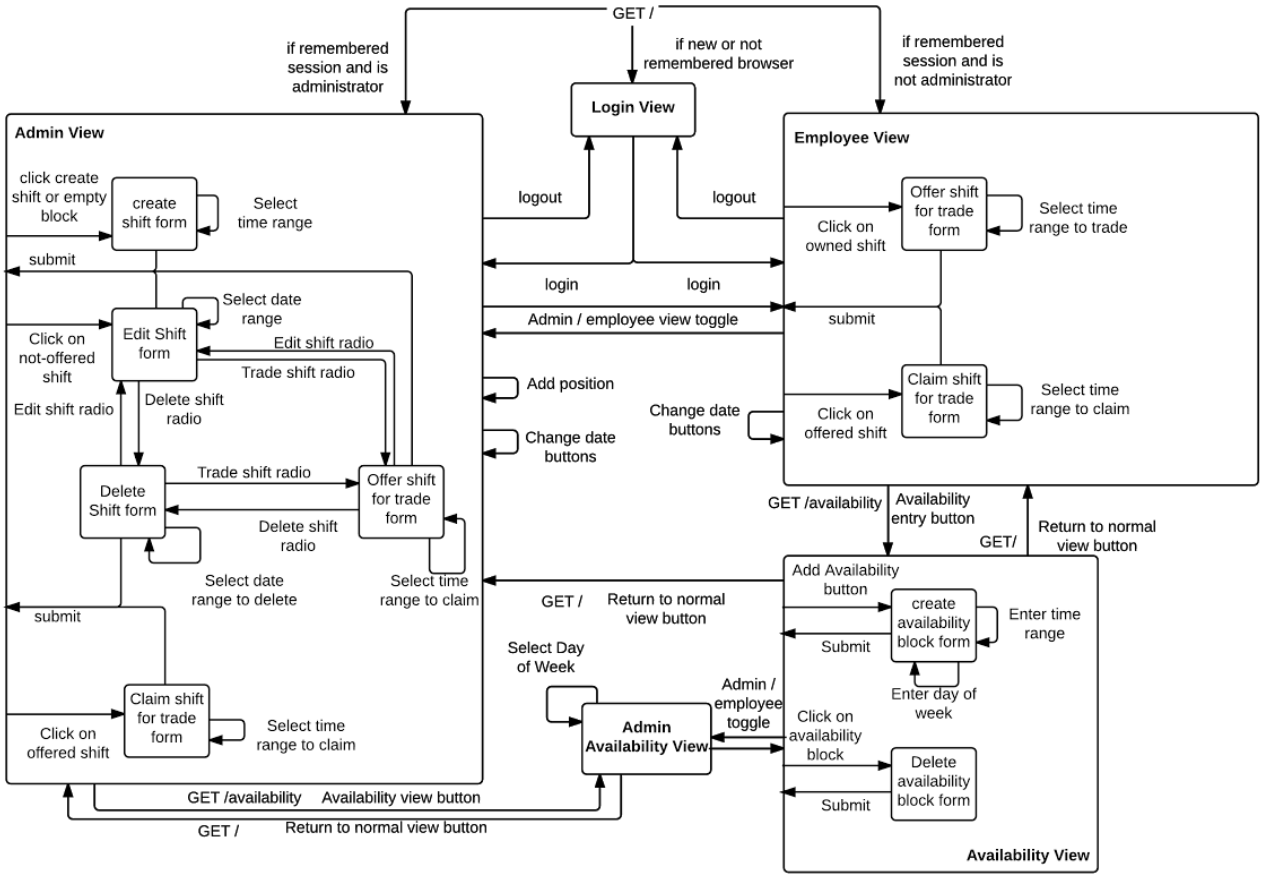
(Chosen) Option: Represent as minutes of a day

This method would require us to create our own Time object, but it would be easily manipulated since it is just a minute count from 0 to 1440. Additionally, it is simple to index a range of time since the start and end times would be distinct

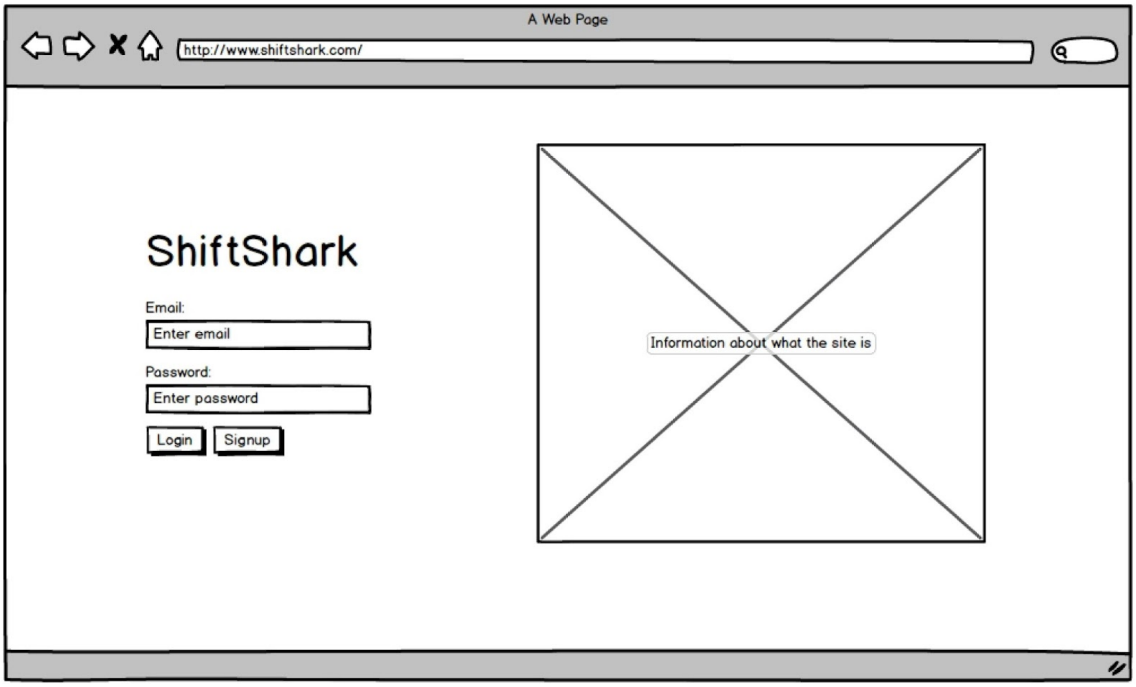
numbers. The only issue would be that it would not give correct representations of time when it comes to the exceptions in day length such as daylight savings time. This unlinking of time and date allows for more modularity and works well with our implementation of shifts.

User Interface

0. Site Flow Wireframe

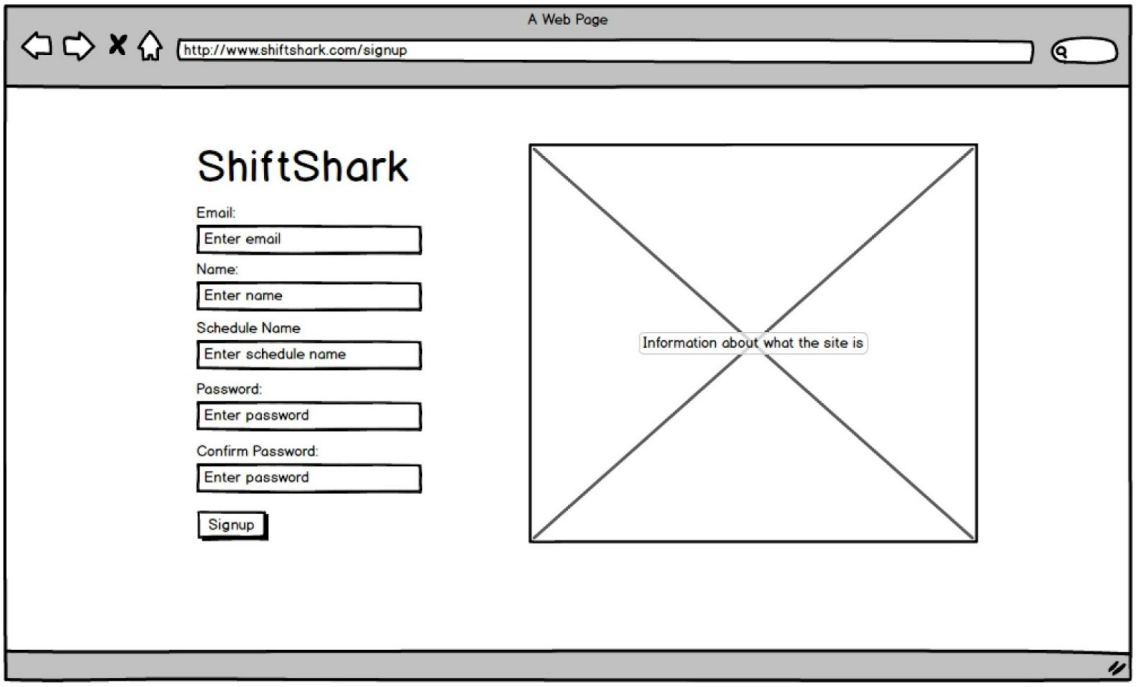


1. Login



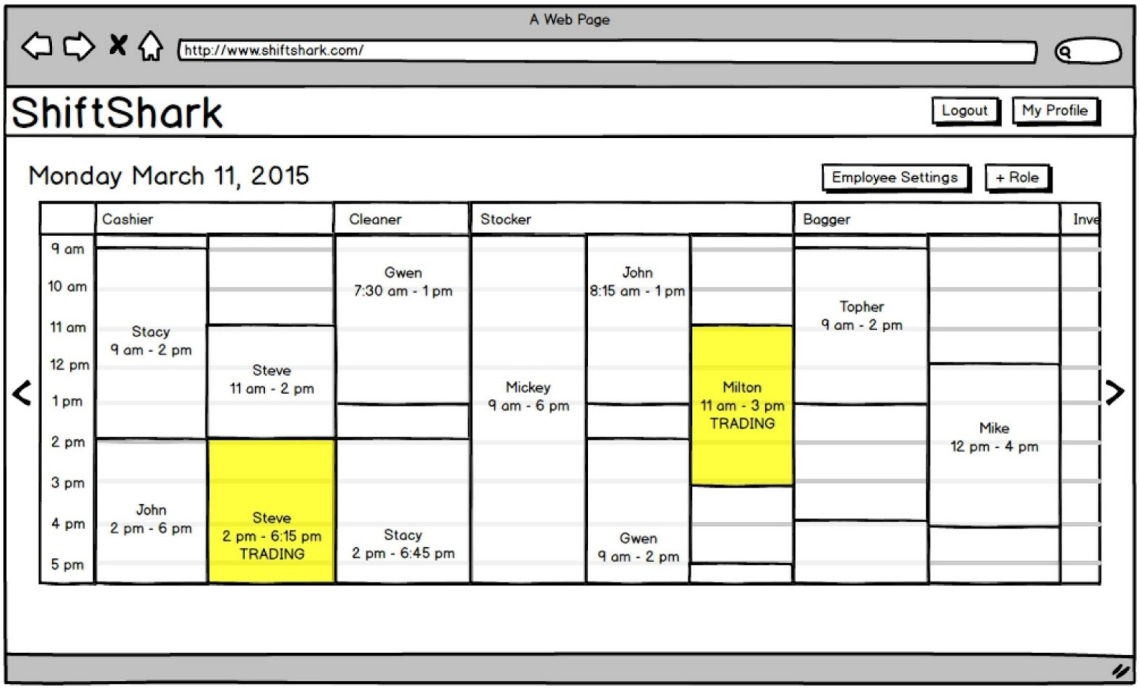
The Login page is the landing page. If they are not authenticated, then they will be brought to this page. They would enter their email address and password and click on the login button to continue. If they fail to authenticate correctly, an error message will appear to indicate that there was an error.

2. Admin Signup



By clicking on the signup button on wireframe 1, they are brought to this page where it prompts the user for their information so that they can create an administrative account and a schedule.

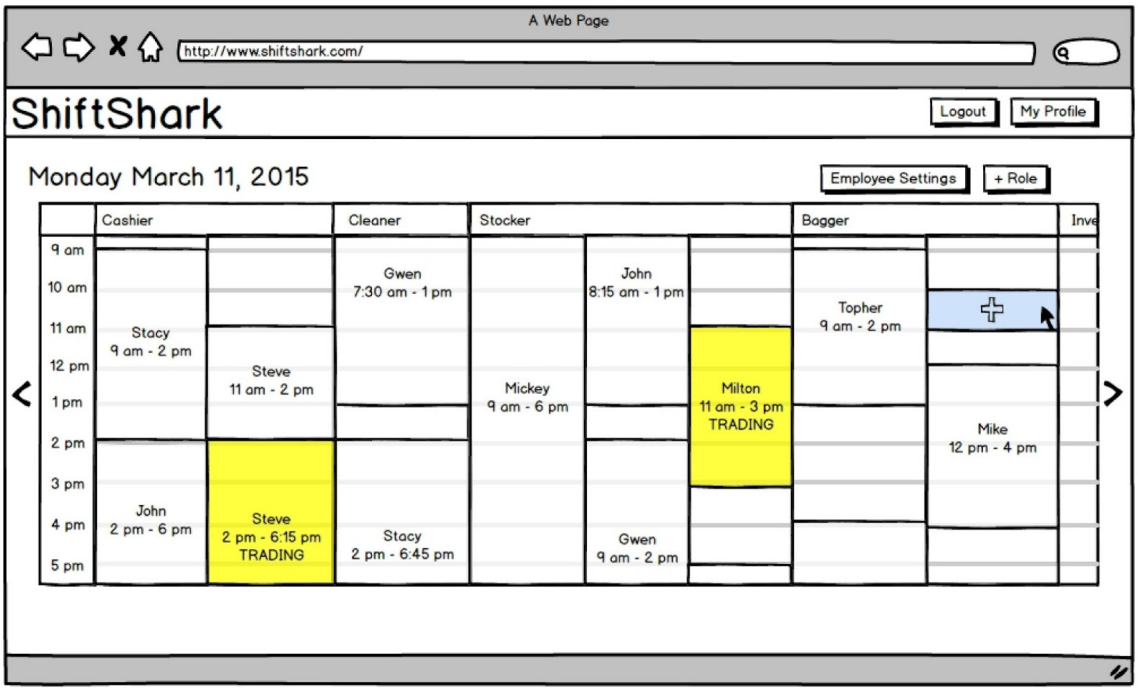
3. Admin Interface



This is reachable if an admin successfully logs in from wireframe 1 or signs up successfully from wireframe 2. The administrative interface shows a single day at a time

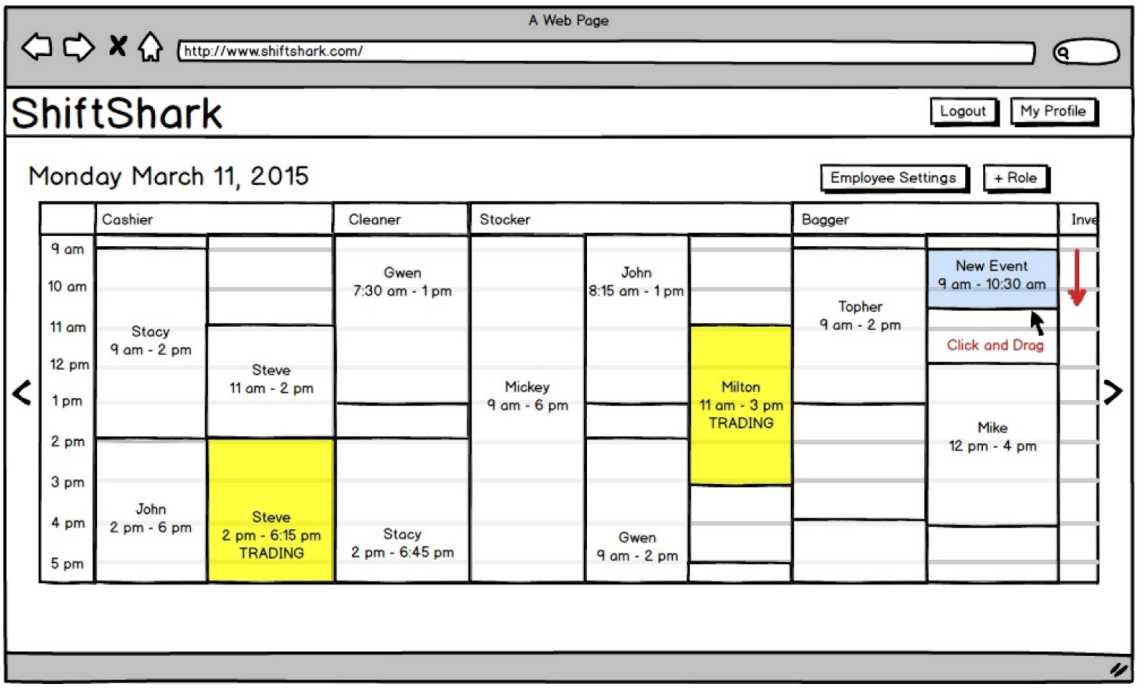
where a day consists of roles (Cashier, Cleaner, Bagger, etc.) and shifts shown within each role spanning a specific time. Unique to the administrative interface are the two buttons to the right of the date: Employee Settings and +Role.

4. Mousing Over Empty Time Slot



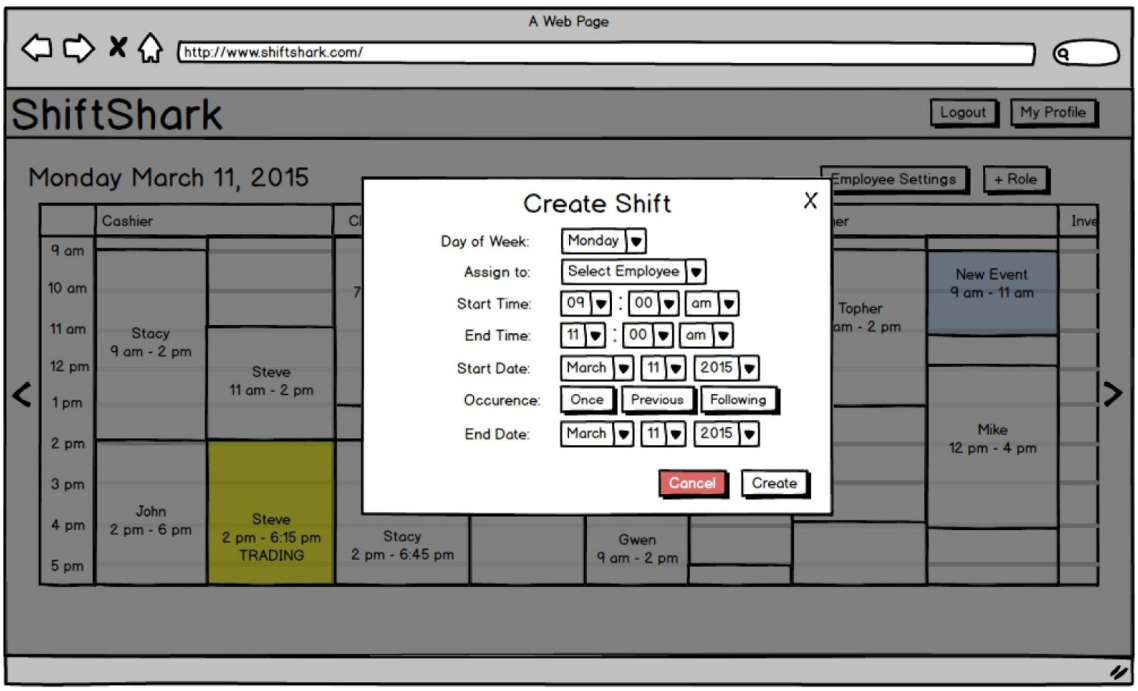
Mousing over an empty time slot in the administrative interface (wireframe 3) creates a highlighted area that indicates that a new event can be created by clicking.

5. Clicking and Dragging Empty Spot



Clicking and dragging on an empty spot in the administrative interface (wireframe 3) will create a new event. It will allow the user to drag over other shifts. If the shifts overlap, then we will use an algorithm that we have already developed to readjust the shifts so that they fit in the least possible amount of columns that do not overlap.

6. Create a New Shift



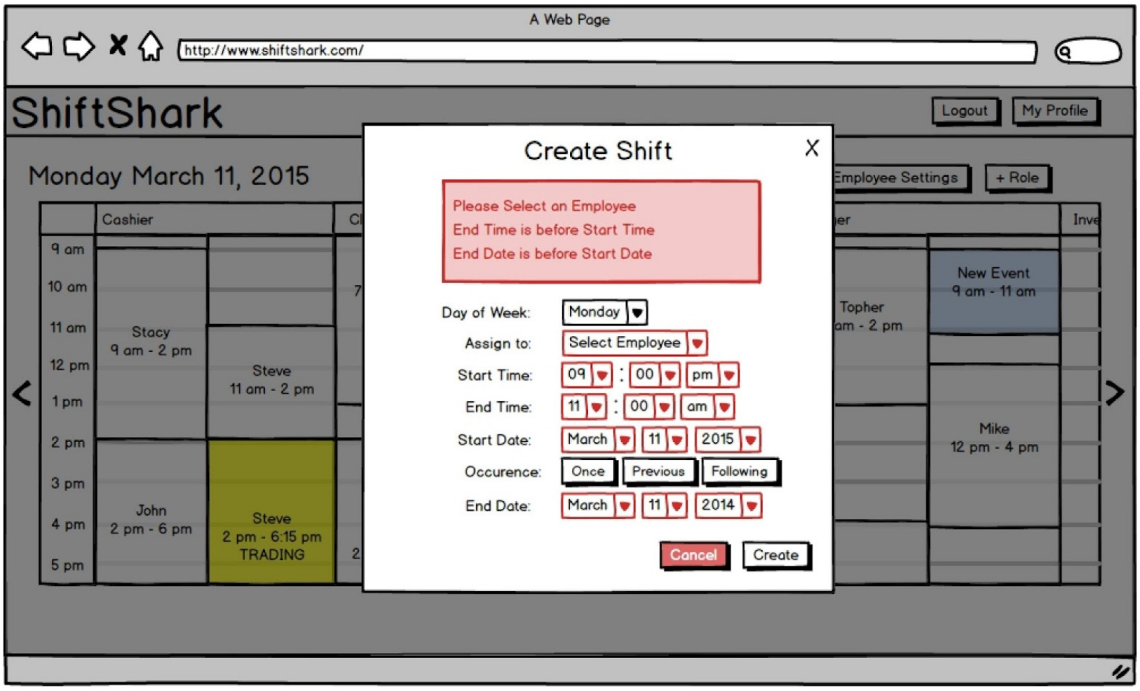
After clicking on the empty spot from wireframe 4 or releasing from the drag in wireframe 5, we have this modal popup. The admin can create a shift by selecting the day of the week, and who to assign it to. The list of people that can have this shift assigned to them include people registered with this schedule by the administrators, nobody (it will be an explicit option stating that it is an open shift that can be claimed), and the administrators themselves since they are employees.

The user can select a start and end time of the shift and a start and end date. If a user selects a start and end date range, then it will only create this shift on the days of the week selected at the top of the form within this range.

We now have the occurrence buttons. These are buttons that will automatically fill out the end date to make this event occur once, to be the last end date this user entered, or to every selected day of the week following the given start date.

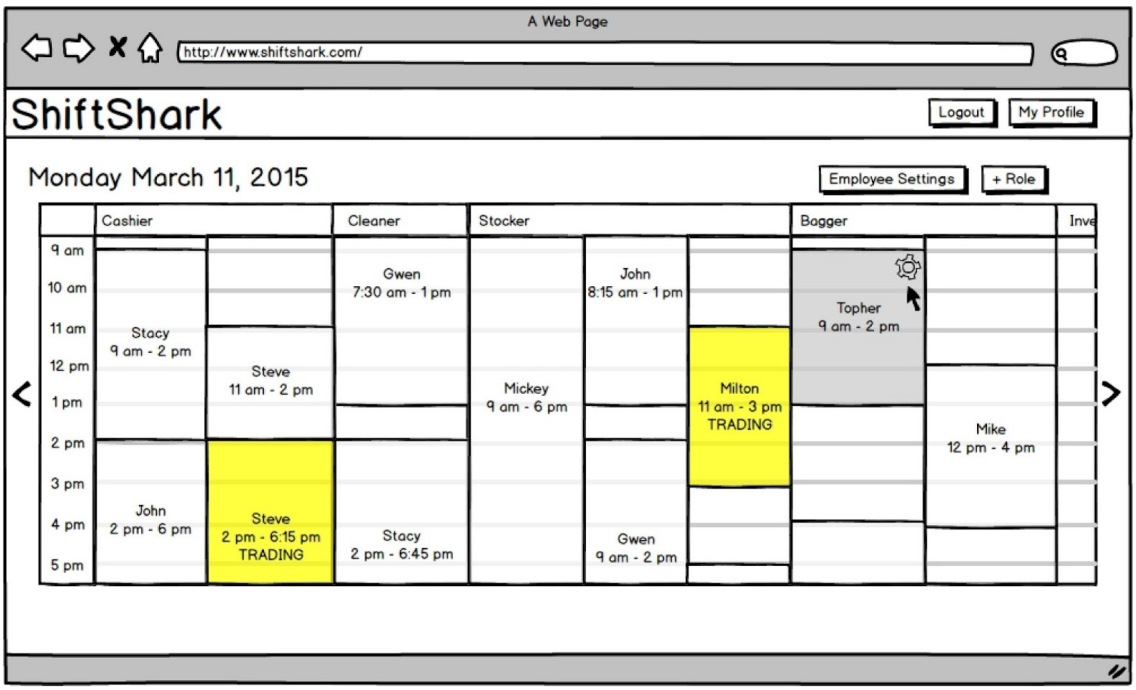
Finally, we have the create button which will create the event and send the user to wireframe 3, and the cancel button which will not create the event and send the user to wireframe 3.

7. Create a New Shift Errors



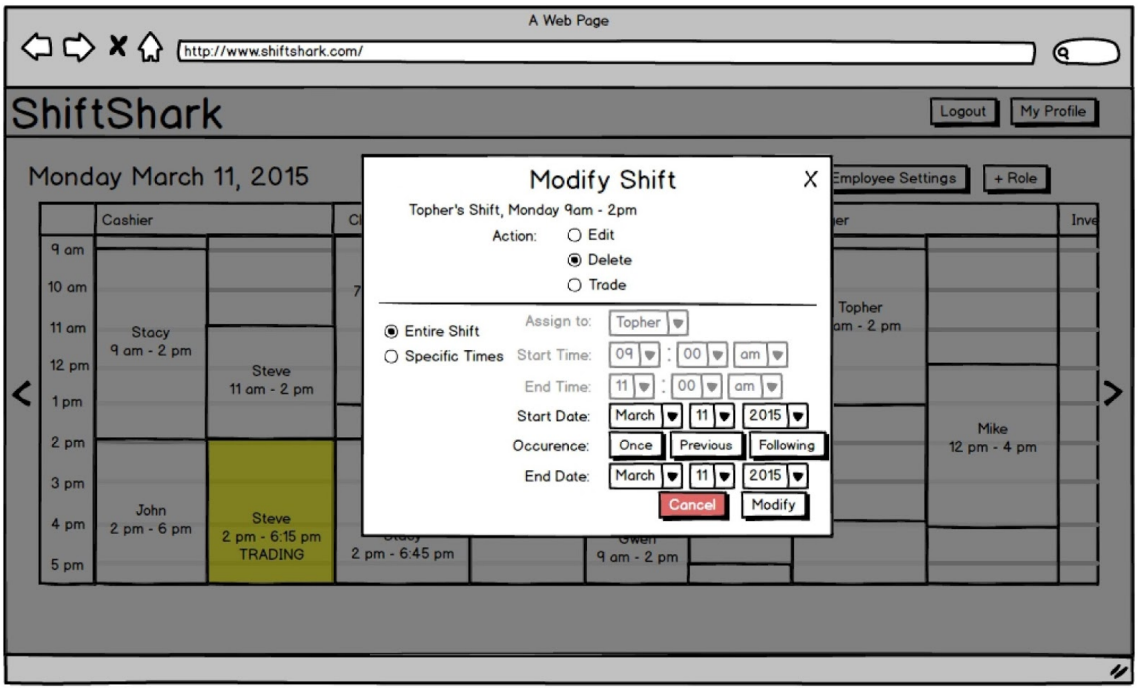
We get to this screen from clicking the create button on wireframe 6. This is how we will show errors within the create shift form for administrators. Possible errors include not selecting an employee (or not choosing to create an open shift) and selecting invalid start and end dates.

8. Admin Hover Over Normal Shift



Reachable from hovering over a normal shift in wireframe 3. This shows a gear icon which will allow the administrator to edit the shift.

9. Admin Edit Normal Shift



Reachable from clicking on the gear icon in wireframe 8. Here the admin is allowed to edit, delete, or put the shift up for trade. There is an option to do this to the entire shift or a specific time. If edit is selected, then the second set of radio buttons will be limited to specific time.

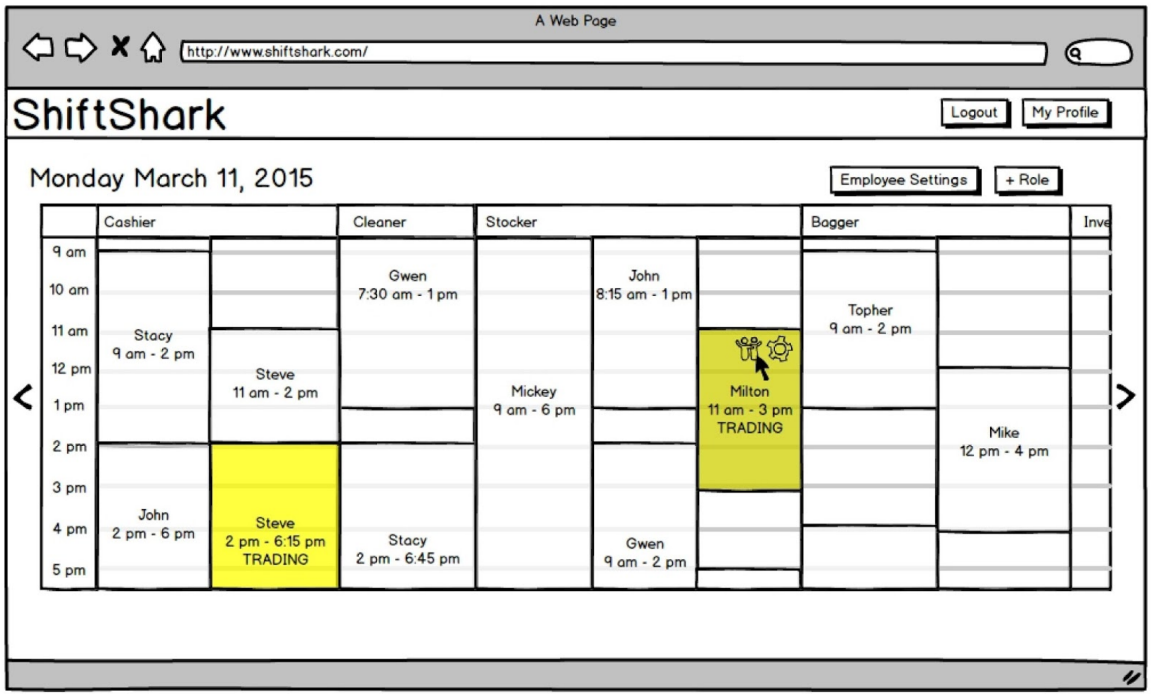
Edit allows the administrator to edit the current shift's start and end times within the given dates. With this interface, the end date of an event cannot be edited. It also allows the reassignment of the shift to someone else.

Delete allows the administrator to delete a shift from a specific time to another specific time and from a date range. This alleviates the issue that the edit option does not allow the administrator to change the end date.

Trade allows the administrator to offer someone's shift up for trade from specific times and date ranges.

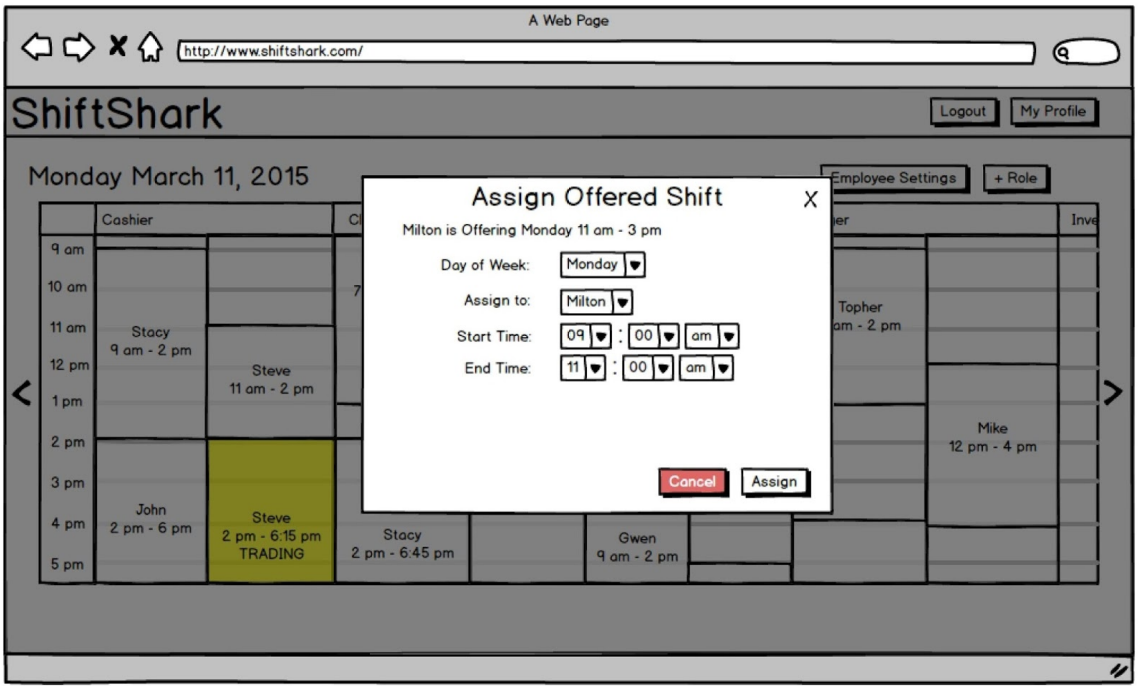
Clicking the cancel button will return the user to wireframe 3 and not do any of these changes, and clicking the modify button will return the user to wireframe 3 and show the changes. An error message will pop up if the user has any issues with the input similar to wireframe 7.

10. Admin Hovering Over a Shift up for Trade



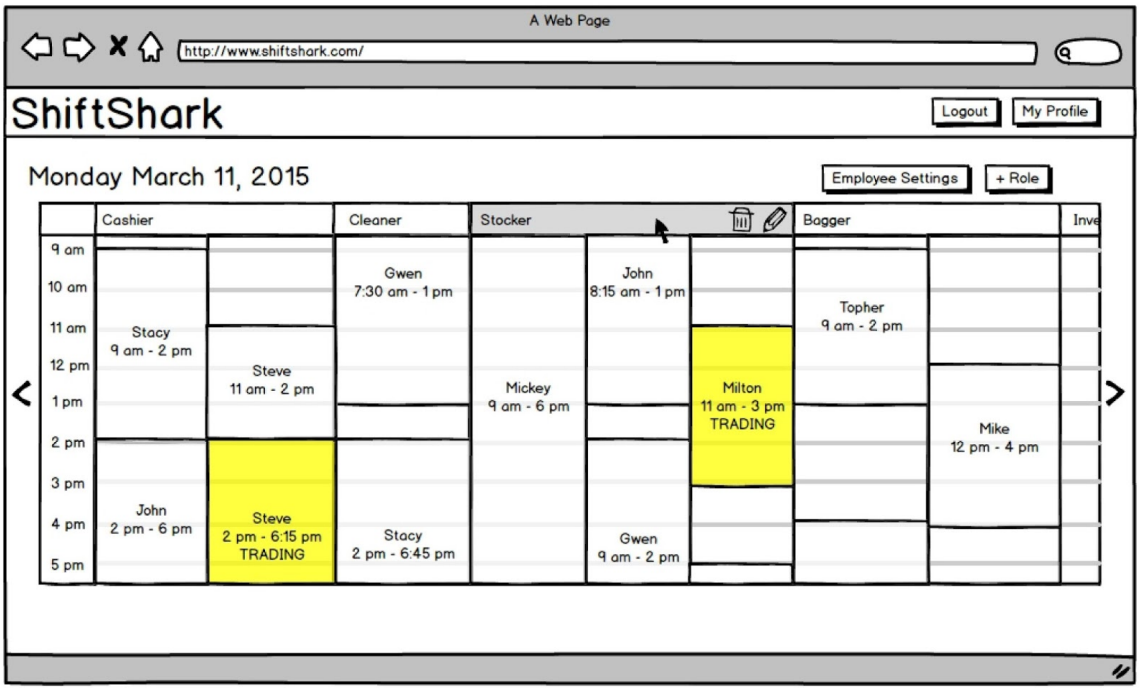
Reachable from an administrator hovering over a shift offered up for trade in wireframe 3. This brings up a gear that will take the administrator to wireframe 9 if clicked and an icon indicating the trade / assign modal.

11. Admin Assign Offered Shift



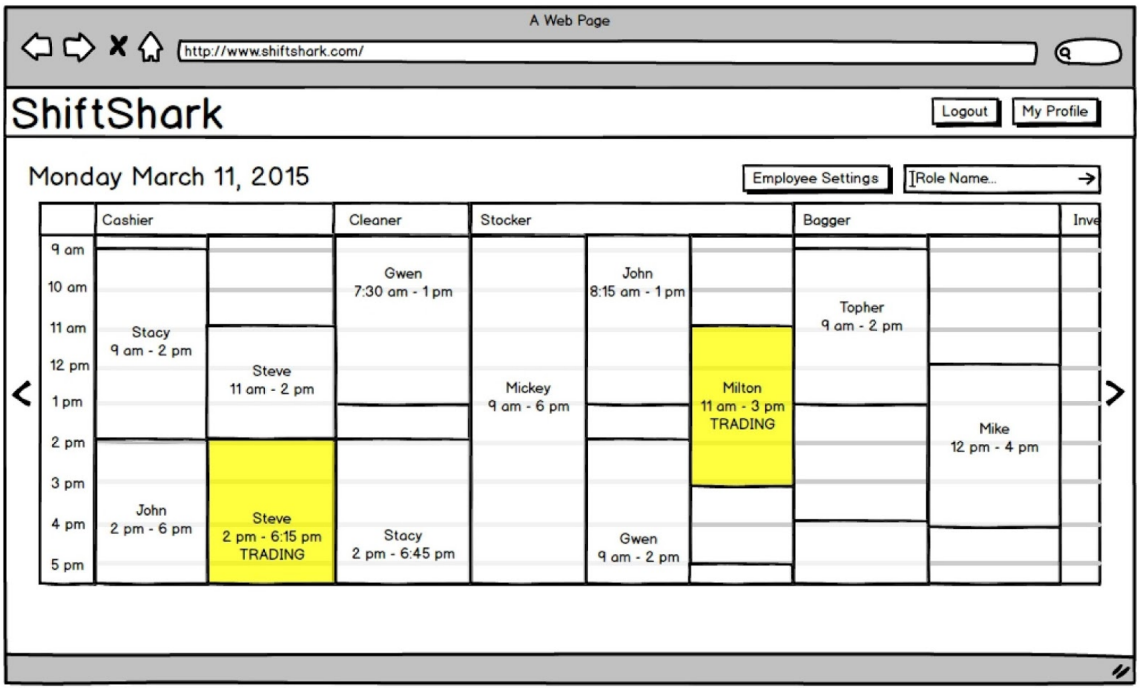
Here the user can assign a shift offered up to trade to an existing user. There will be error checking similar to wireframe 7. Clicking on cancel or assign will return the administrator to wireframe 3.

12. Admin Mouse over Role



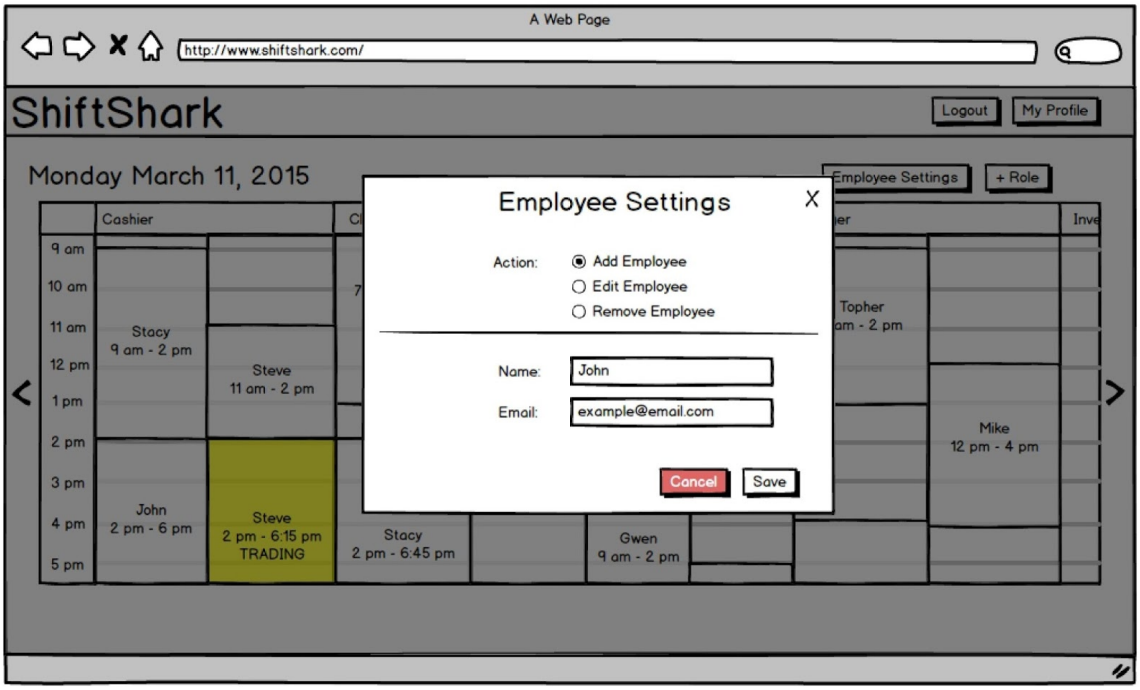
If an administrator mouses over a role in wireframe 3, then two icons will pop up. An icon indicating a deletion of a role which will delete the role and subsequent columns associated with it. There will also be an edit icon that will allow the user to change the name of the role (possibly with overlaying a textbox). All of these actions will return the user to an updated wireframe 3.

13. Adding a Role



Reachable from clicking on the +Role button in wireframe 3, the role button turns into a textbox with a submit button. Upon hitting enter or clicking the submit button after typing a role name, it will create a new role with an empty column with the given name. This returns to wireframe 3.

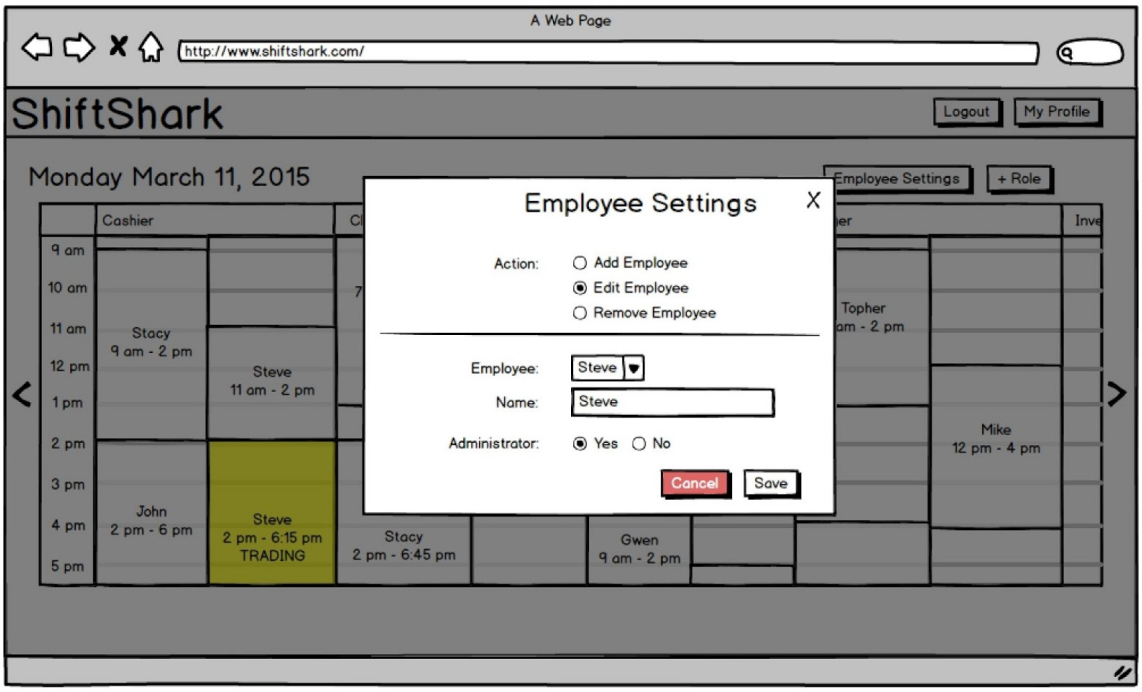
14. Employee Settings: Adding an Employee



Reachable from clicking on the Employee Settings button in wireframe 3 or “Add Employee” radio buttons in wireframes 15 and 16. This allows the administrator to add employees to the schedule. The administrator types in a name and the email of the employee, and then hits save to send the invitation email to the user. This will also create an entry despite the invited user accepting or not, because this will allow the administrator to immediately create shifts for the new user.

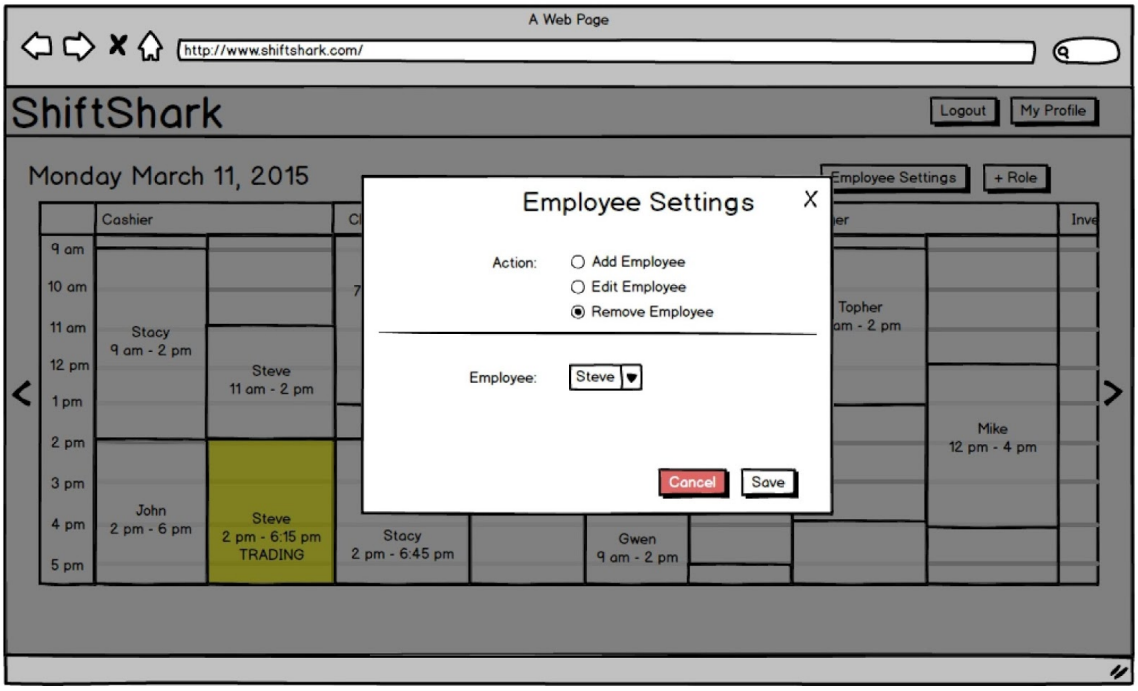
There will be error checking with error messages like in wireframe 7. Clicking on save or cancel will return the user to wireframe 3.

15. Employee Settings: Editing an Employee



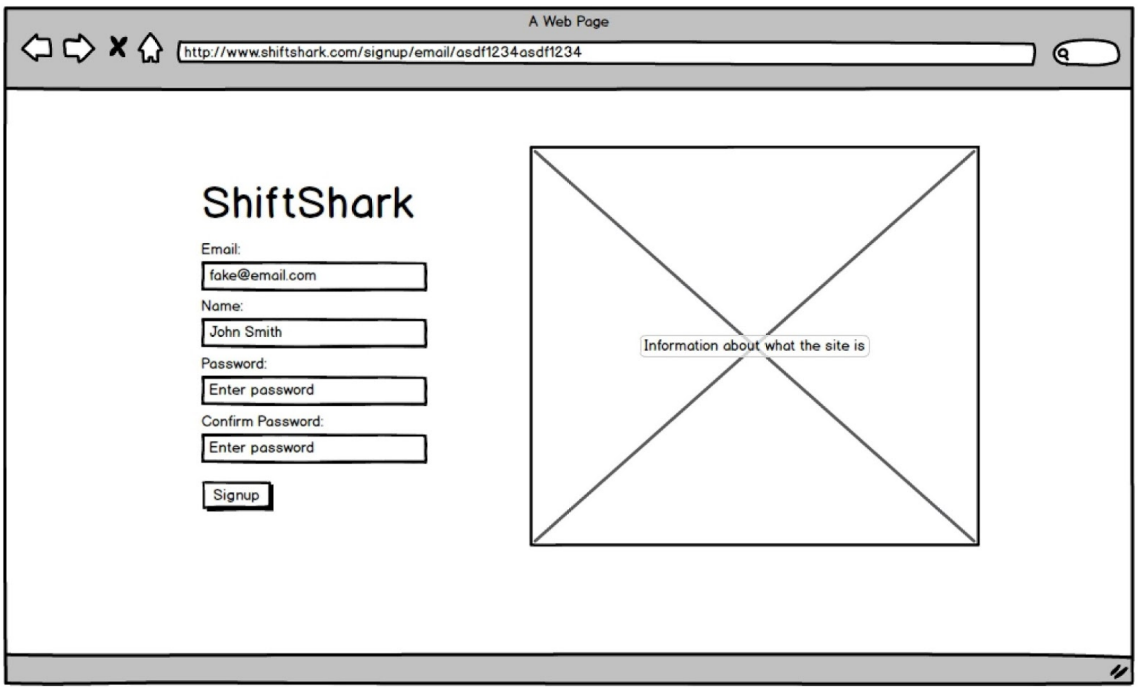
Reachable from clicking on the “Edit Employee” radio button in wireframe 14 and 16. Here the administrator is able to change the name and set administrative privileges of employees. All administrators can change the administrator privileges of other administrators. Clicking on cancel or save will return the user to wireframe 3.

16. Remove Employee



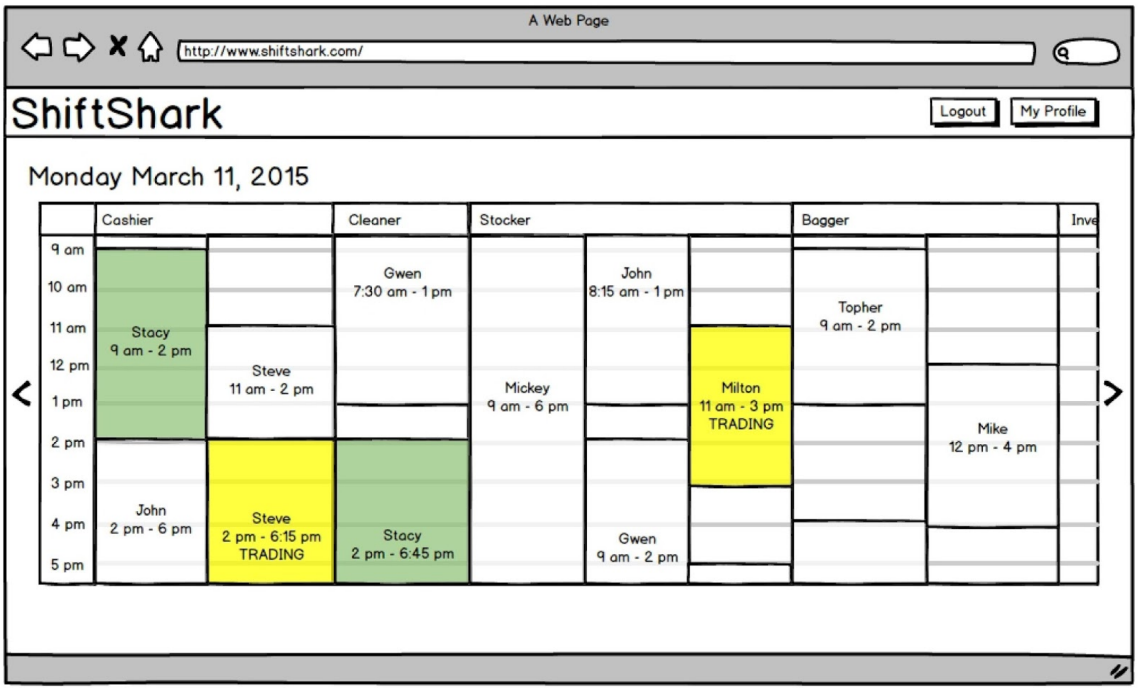
Reachable by selecting “Remove Employee” from wireframes 14 and 15. This allows administrators to remove employees from the schedule. The save and cancel buttons both return you to wireframe 3. This will also delete any shifts held by that employee.

17. Employee Signup



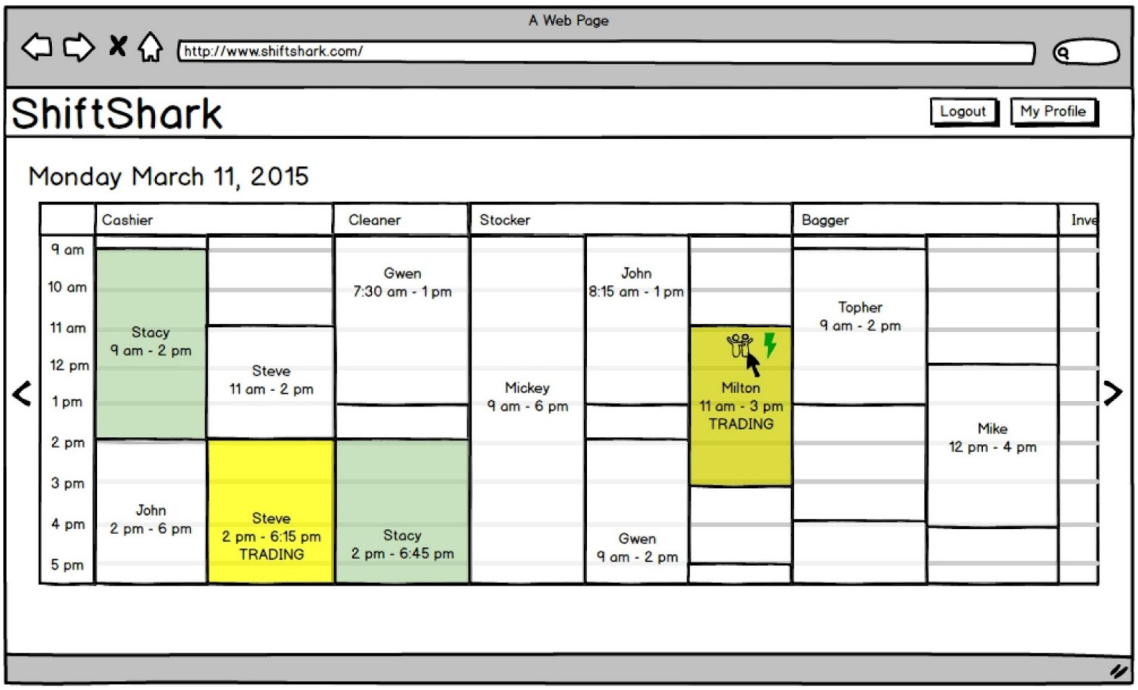
This is only reachable by referral link that is sent out to employees in an email after submitting wireframe 14. Most of this is prepopulated and can be changed by the employee signing up for ShiftShark.

18. Employee Schedule View



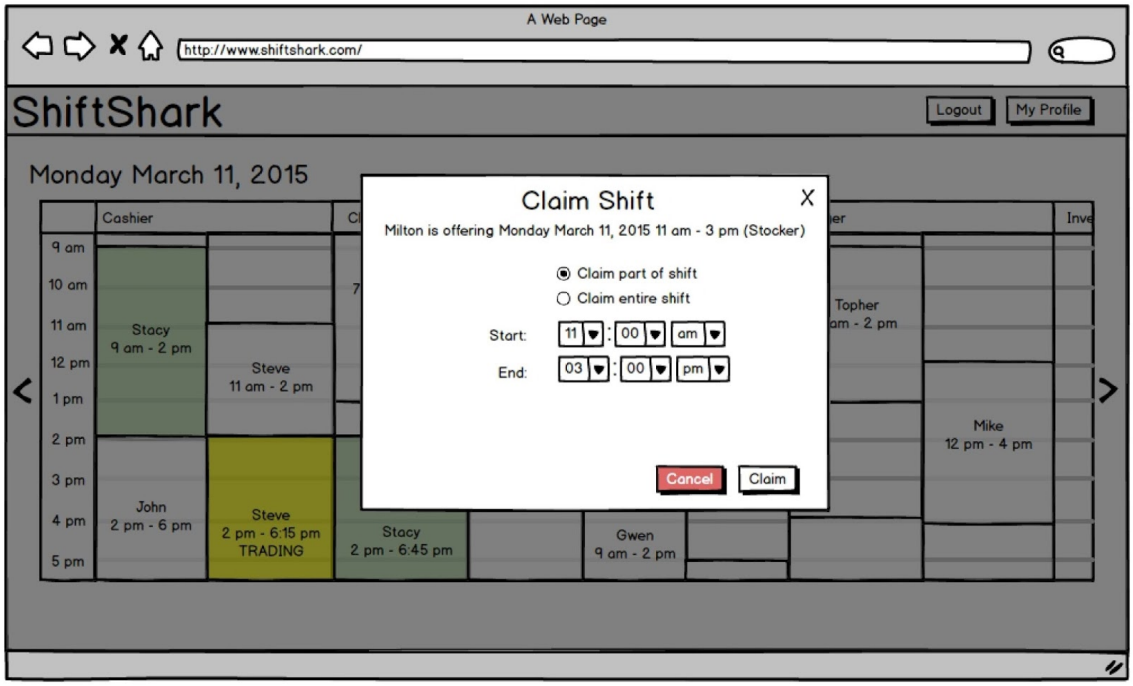
Reachable from successfully authenticating with wireframe 1 and clicking on login, or by filling out the form in wireframe 17 and clicking on signup. This is the employee view of a schedule. Their own shifts are shown in green, and shifts offered to trade are shown in yellow.

19. Employee Hover Trade



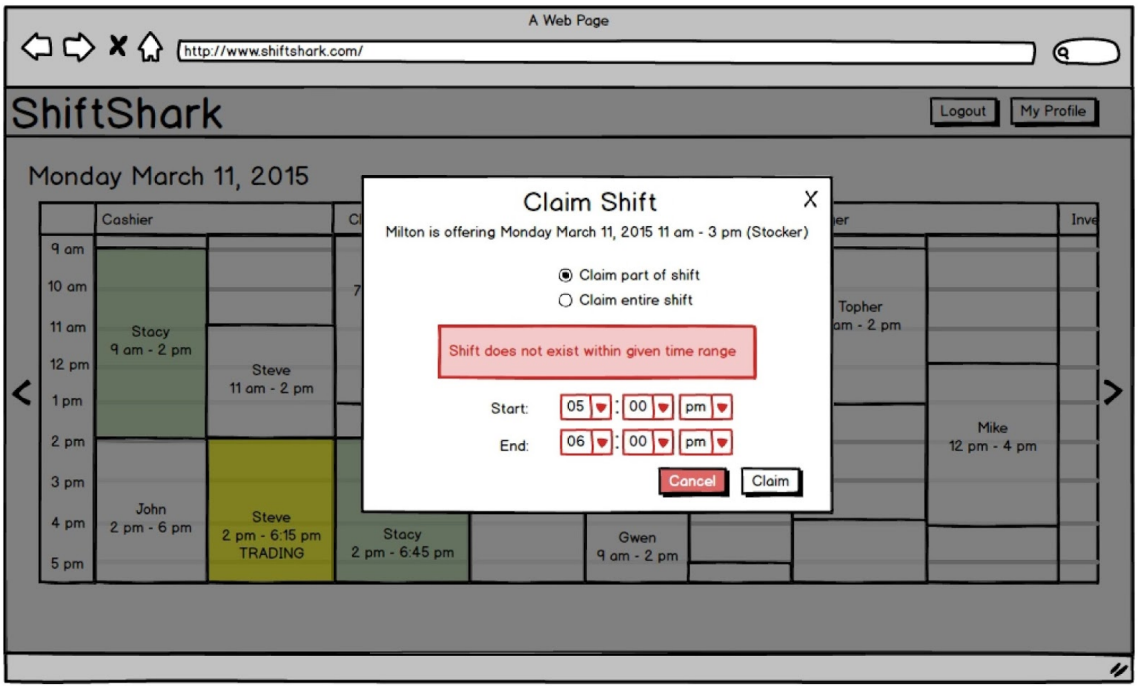
Reachable by hovering over a shift offered to trade in wireframe 18. A normal employee sees two icons appear on the shift. A lightning bolt which would automatically claim that entire shift for only that day. The other icon is the trading icon.

20. Employee Claiming



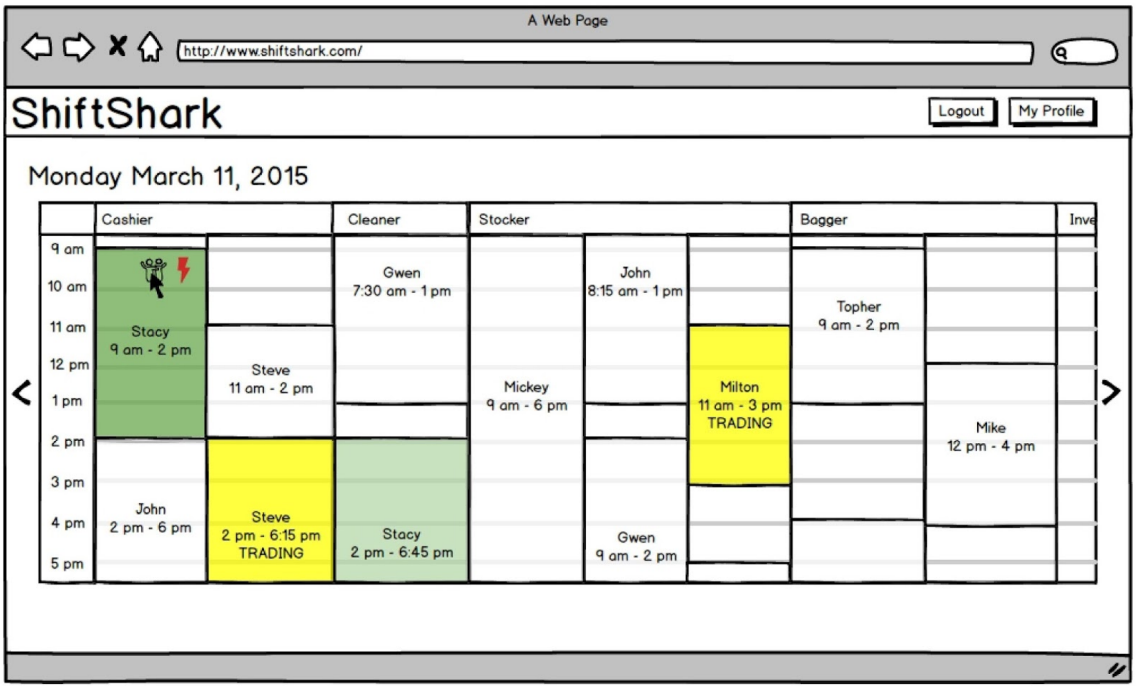
Reachable by an employee clicking on the trading icon in step 19. This allows the user to claim an entire shift or part of a shift. Clicking claim or cancel may lead to wireframe 18 or 21.

21. Employee Claiming Error



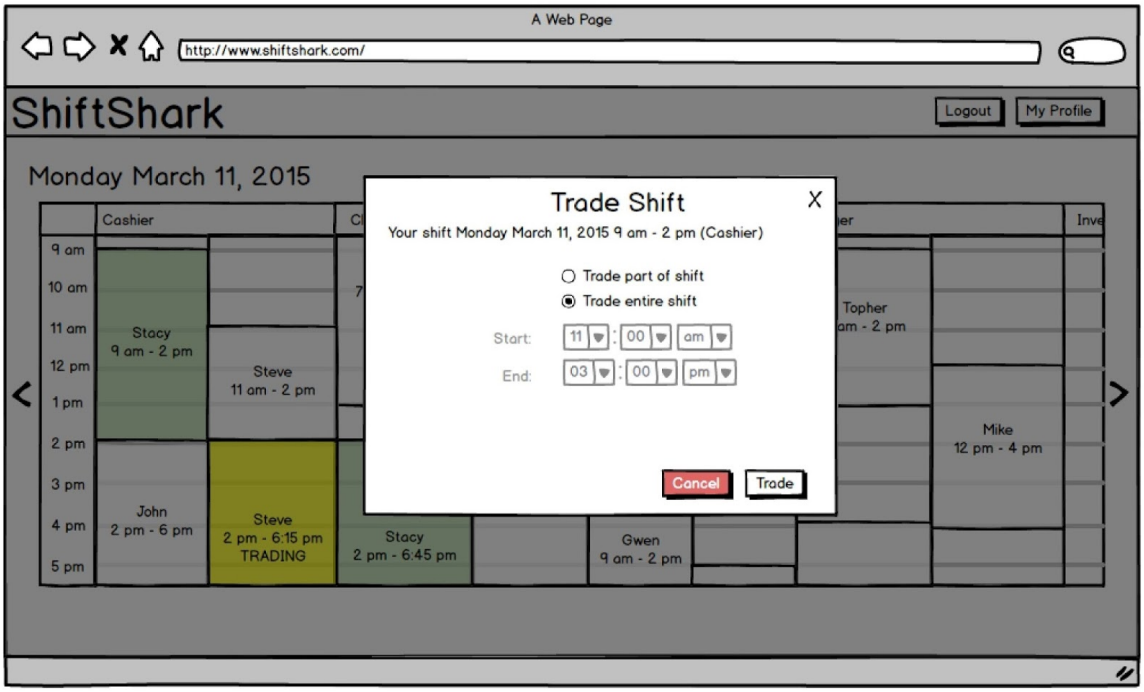
Reachable from clicking on the claim button and filling the form out wrongly in wireframe 20. This is an example of an error that can come up with this implementation where a start and end time Is chosen outside the range.

22. Employee Hover Self



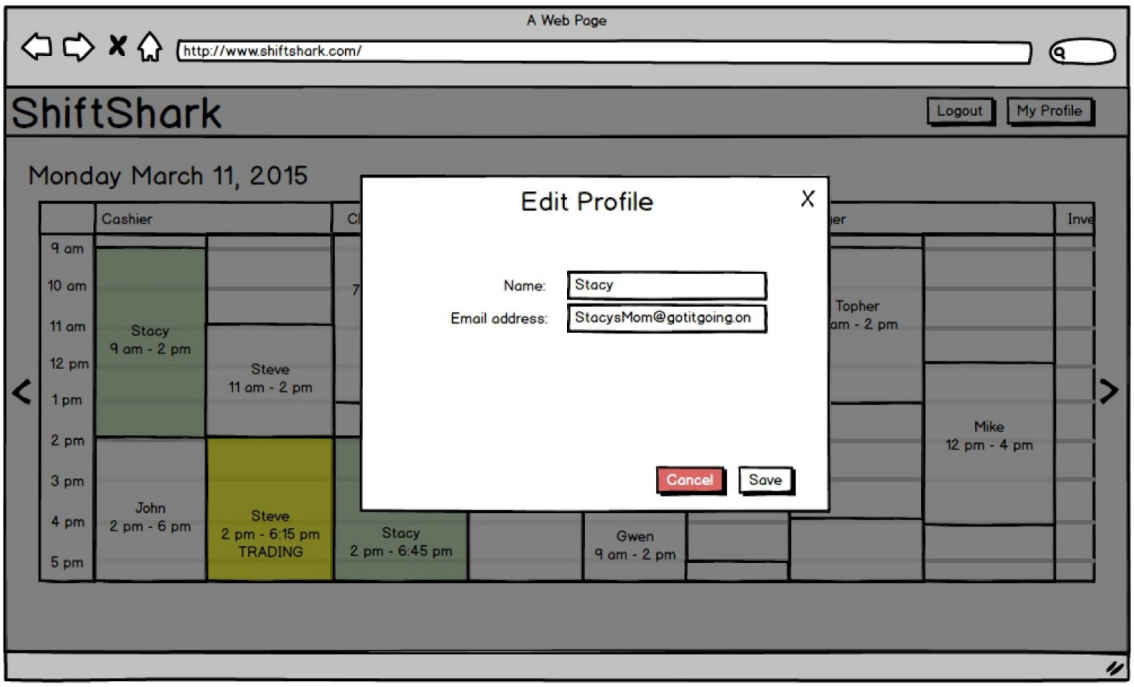
Reachable by hovering over a shift assigned to the current user in wireframe 18. A normal employee sees two icons appear on the shift. A lightning bolt which would automatically trade that entire shift for only that day. The other icon is the trading icon

23. Employee Trade



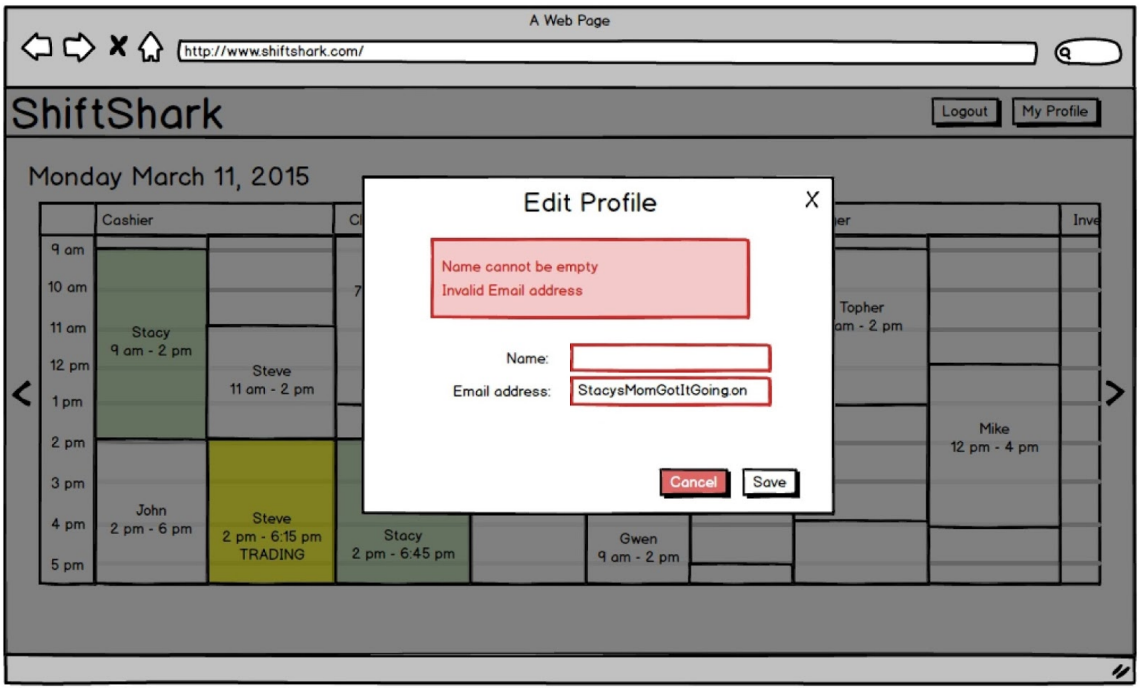
Reachable by clicking on the trading icon in wireframe 22. This page allows the employee to put one of their shifts up for trade. Clicking on Trade or Cancel will move the user to wireframe 18 or show an error if there was one.

24. Edit My Profile



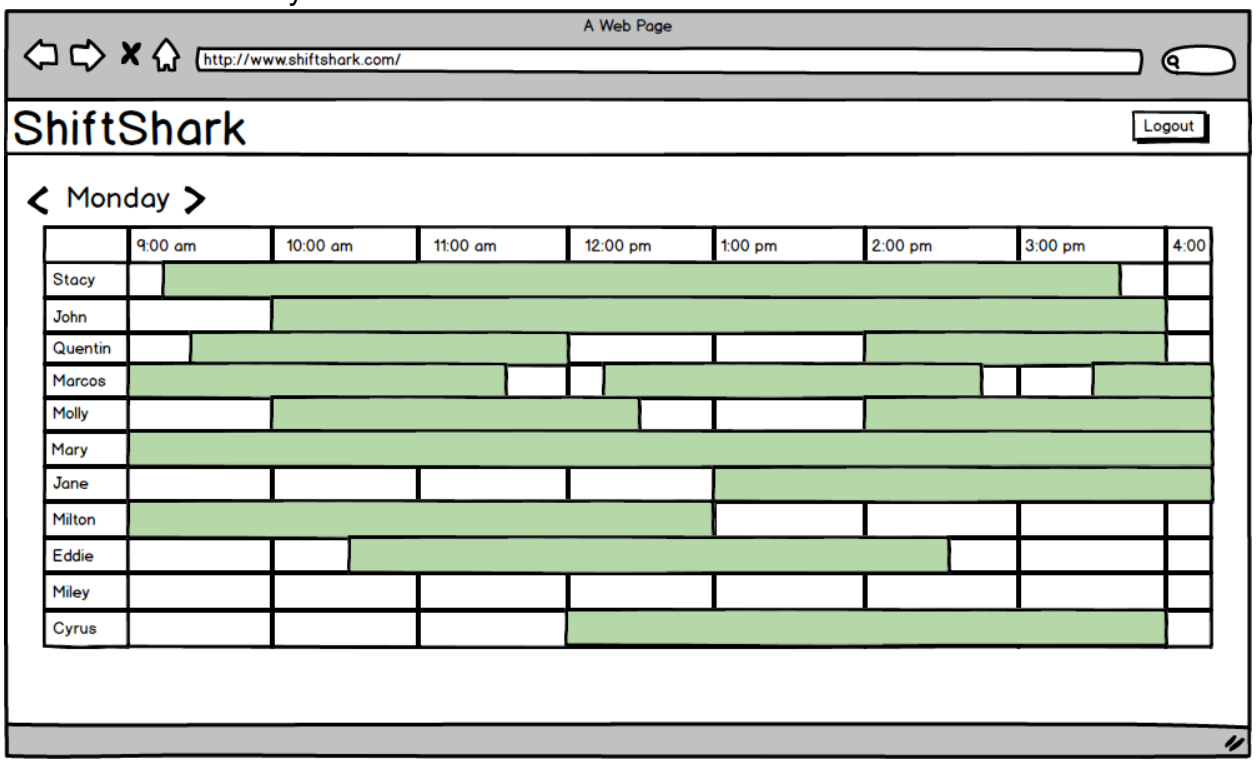
Reachable by clicking on the My Profile button in wireframes 3 or 18. This allows the users to change their names or their email addresses associated with the schedule. The save and cancel buttons will move the user to wireframe 18 or show an error if the form was filled out incorrectly.

25. Edit My Profile Error

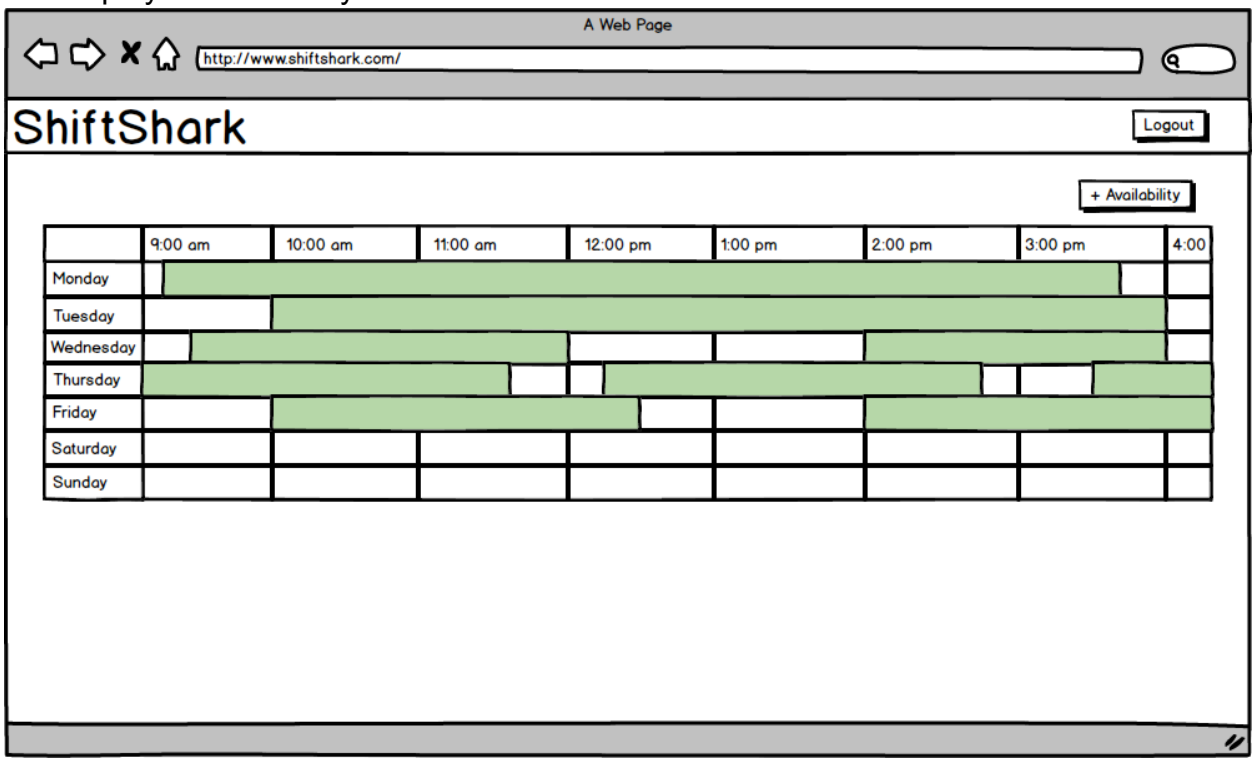


Reachable by filling out information incorrectly and hitting the save button in wireframe 24. This shows some possible errors that can be associated with editing a profile.

26. Admin Availability View



27. Employee Availability View



28. Employee Set Availability

29. Employee Delete Availability Block