6.170 Project 4: Zhift

DESIGN DOCUMENT BY **ANJI REN | DYLAN JOSS | LILY SEROPIAN | VICKY GONG**

Overview

Zhift is an online application that helps organizations manage who is working and when by facilitating scheduling shifts for employees with changing availabilities.

Background / Target

There are many organizations in which multiple people can work the same roles, but often different hours. For example, in a restaurant, different roles could be cooks and waitstaff, where employees who are cooks and waiters/waitresses work shifts that change weekly. Once shifts are assigned to workers, workers are responsible for the shift. However, employee availability may change unpredictably on a weekly basis, and employees may be required to call either a manager or other employees in order to change their shifts.

Motivation

The process of adjusting weekly schedules is often cumbersome (e.g. restaurant waitresses, reception desk staffing, EMTs, etc). Zhift aims to facilitate adjusting weekly schedules by automating much of the shift change request process.

Purposes

- **Enable employees to adjust their schedules on a weekly basis to accommodate changes in their availabilities.** Life happens - illnesses, celebrations, and jury duty are just a few situations that can cause an employee's availability to change for the upcoming week. Zhift aims to make it easy for employees to balance their work schedules and their personal lives by providing an interface for employees to find other employees to take responsibility for the shifts they cannot make.
- **Aid managers in ensuring all shifts are filled even when individual employees' availabilities change.** If an employee has an unexpected conflict with his scheduled shift and cannot find someone else to take it, that shift may go unworked. Zhift helps keep managers informed of these situations so they can step in and get the unclaimed shift filled, or make alternative plans if no one is available.
- **Keep track of who is currently responsible for each shift.** Shifts can go unfilled if an employee makes an agreement to take someone's shift, and then reneges on that agreement. Zhift adjusts the schedule to reflect such agreements in order to avoid employees forgetting about the shifts they claim, and to assign culpability in the case that a shift goes unfilled/unworked.
- **Let managers and employees know in real-time of schedule changes.** Organizations run most smoothly when everyone knows what's going on - especially

the people in charge. Zhift keeps everybody up-to-date so there are no unpleasant surprises when people turn up to work.

Target Users & Stakeholders
Managerial staff and employees of small businesses and organizations.

Existing Solutions

- **Manual.** Some organizations use a paper and pencil approach to keeping track of their employees' shifts. The process surrounding this solution methodically involves a manager in charge of bookkeeping regularly scheduled shifts and changes in who is working what shift. These changes are communicated from employees to managers in person, by phone or email, and managers are responsible for mediating any changes to ensure the organization still meets its staffing needs.
- **In-house custom applications.** Some organizations have custom applications that are specific to the needs of the organization. These solutions can be expensive to develop and inflexible.

System Design & Details

Context Diagram
See `Context Diagram.pdf.`

Concepts

- **Organization** : All users of Zhift only ever interact with anything within their organization. Organizations that use Zhift are generally small (< 20 people), and have part-time and full-time employees whose availabilities change frequently.
- **Manager** : a person who assigns shifts. Managers are administrators for an organization.
- **Employee** : a person who is assigned to shifts. Employees work for an organization.
- **Template Shift:** a contiguous block of time during which an employee works every week (for example, Wednesdays from 2-4pm).
- **Shift** : a contiguous block of time during which an employee works on a specific week (for example, Wednesday, November 12, from 2-4pm). When employees make shift changes through Zhift, they change the person responsible for the shift, not for the template shift.
- **Swap** : an exchange of shifts between employees. Swaps are one-time-only, and must be for two shifts in the same week.
- **Offer** : to give up a shift. Employees offer shifts when they cannot work them and do not want to take on a different shift.
- **Claim** : to take an offered shift. Responsibility for a shift changes when an employee claims an offered shift.
- **Responsibility** : to be responsible for working a shift or finding someone else to work it. Any shifts assigned to an employee are that employee's responsibility.

Responsibility for a shift changes when a swap is accepted or a claim occurs.

- **Role** : a type of job an employee performs. Example roles include waitress, hostess, cashier, chef, etc.
- **Schedule** : a schedule for a given time period and a given role, e.g., all the cashier shifts for next week.
- **Record** : a summary of a change that occurred regarding shift ownership in a schedule.

Data Model

See `Data Model.pdf`.

Data Design

See `Data Design.pdf`.

Data Schema

TemplateShift:

- `dayOfWeek: String`
- `start: Time (hr of day)`
- `end: Time (hr of day)`
- `responsiblePerson: Employee`
- `schedule: Schedule`

Normal Shift:

- `Everything in TemplateShift`
- `dateScheduled: Date`
- `upForGrabs: Bool`
- `upForSwap: Bool`

Schedule:

- `role: String`
- `org: String`

Record:

- `schedule: Schedule`
- `content: String`

Swap:

- `shiftUpForSwap: Shift`
- `shiftOfferedInReturn: Shift`
- `schedule: <ScheduleId>`

User:

- `email`
- `org`
- `password`
- `username`

Manager:

- `[none]`

Employee:

- `schedule: Schedule Id`

Organization:

- `name`

Behavior

Security Requirements

A primary security concern of Zhift is maintaining the integrity of an organization's shift schedules. That is, schedules should reflect the changes that are made by authorized users (i.e. managers or employees linked to the organization). Furthermore, only managers of the organization should be able to add or remove employees - employees should not have the same "root" privileges as the managers. Lastly, individuals not associated with the organization should not be able to make any modification to the organization.

Addressing Security Requirements

Sanitize all text input

To defend against potential injection attacks, we plan to sanitize text input, including, but not limited to, users' usernames and passwords, emails sent to users, and text related to the initialization of organizations or user accounts. This will also mitigate the possibility of XSS attacks.

Password encryption

To store passwords more securely, we plan to use the bcrypt hashing algorithm. Passwords will not be stored in plaintext.

Authentication

To ensure that only users of the organization may make the various changes to the organization's shift schedule, we plan to check for user login credentials before executing the requested action. Both the view and controller components will perform these checks, so as to ensure that the correct information is displayed to the user and that the user can only take the actions for which he/she is enabled. This will also mitigate the possibility of XSS attacks.

Authorization

By having separate account types for employees and managers, we can restrict the actions that may only performed by managers to those with manager accounts.

Notifications of Schedule Changes

Managers of an organization receive notifications about schedule changes both via the Zhift Dashboard and email. Managers can view pending and completed schedule changes and take action if a particular change seems suspect.

Threat Model and Assumptions about Attackers

Possible Adversaries

User of a competitor Organization X who wishes to sabotage Organization Y's schedule - our account authentication schema ensures that only users of Organization Y may make changes to Organization Y's schedule/other data.

Disgruntled employee of Organization X wishes to sabotage the schedule of Organization X - employees are only enabled to make changes to their own schedules. Should an employee gain the credentials of another employee (e.g. through social engineering, keylogging) and make changes to that employee's schedule, the manager would be notified of the suspect changes and could take action.

Disgruntled manager of Organization X wishes to sabotage the schedule of Organization X - other managers would be notified of the suspect changes and could take action.

Individual wishes to conduct DDoS or similar attack on the site - we plan to take advantage of OpenShift and the security features that it and Amazon EC2 provide (https://www.openshift.com/policy/security)

API

See `API.pdf`.

User Interface

Wireframes

See Wireframes directory

User Flow

See `User Flow.pdf`.

Design Challenges

What Gets Created First - Organization Or Manager?

- Organization first
  If a user creates an organization first, and then exits the initialization process before adding themselves as a manager for that organization, they cannot access that organization again and there's an orphan organization floating around in the database that will never be used.

- Manager first
  If a user creates a manager account first, and then exits the initialization process before creating an organization, there's now a user that is not associated with an organization. A user cannot do anything with Zhift without having an organization, so this scenario is useless.

- Organization and manager get created together
  Organizations must have at least one user and users must belong to at least one organization, so the initialization workflow will create an organization and a manager account in the same step.

Should Employees Have Roles?

- No roles
  Originally, Zhift did not have a concept of roles. The only differences between users were whether they were employees or managers. This resulted in a simple, clean design. However, without a concept of roles, things would be allowed that would not make make sense. For example, an employee who works as a cashier would be able to swap shifts with an employee who works as a chef. Having all employees on the same schedule would also complicate the user experience. When managers are assigning shifts, if they are viewing a schedule with all of the employees on it, figuring out if they need to schedule more cashiers might be difficult because of the cognitive overload of having everyone on the same schedule. For employees looking to pick up additional shifts, seeing all available shifts is much less useful than seeing only the shifts they are qualified to work. Thus, Zhift required a concept of roles.

- Multiple roles
  An alternative model is that an employee can have one or more roles. This multiplicity of roles was inspired by Professor Jackson's example where Whole Foods employees who are scheduled to stock shelves will switch to being cashiers if the lines get too long. Having roles eliminated the problem from Option 1, but an individual being able

to have multiple roles introduced new problems. The design got much more complicated, especially in terms of permission access and user interface design. For example, a stocker/cashier would be able to claim an offered-up cashier shift, but a cashier would not be able to claim a stocker/cashier shift, and the logic gets even more complicated as you add more roles to a single person.

- <u>One role per user</u>
  An organization having an employee who fills multiple roles is a rare enough case that it does not warrant complicating the design. Zhift now supports employees with exactly one role - no more, no less. This fixes the problem of employees with different roles swapping shifts, while avoiding the design nightmare of multiple roles. If an organization really needs to have an employee with multiple roles, they can either be creative with their role-naming (e.g. create a new role whose name is stocker/cashier), or create accounts for that employee for each of their roles (since accounts are identified by organization, username, and role).

Should Zhift Support Scheduling Managers?

- <u>Yes, build manager scheduling separately into manager accounts</u>
  Duplicating functionality is almost always bad. Zhift can already schedule employees, no need to write it differently for managers.

- <u>Yes, make “Manager” an automatic role.</u>
  Creating an automatic role of “Manager” would allow managers to be scheduled the same way employees are scheduled. This could lead to some weird permissions situations however, since managers also have the ability to change schedules without going through the swap/offer/claim process. Zhift is designed to allow employees to change their shifts without needing to go through the managers and without messing up the schedules. Allowing managers to schedule themselves in this way defeats those purposes.

- <u>No, managers should not have schedules for themselves.</u>
  Zhift is designed for the part-time employees of an organization, not the full-time managers, and thus supporting manager scheduling really isn’t necessary. For the sake of scope and simplicity, Zhift will not support it. If an organization really needs to schedule their managers, they can use the same work-around as for having employees with multiple roles: create employee accounts for the managers and do the scheduling for those accounts.

How Many Employees Can A Shift Have?

- <u>Allow shifts to have as many employees as are working at that time</u>

One way to represent shifts is to have each shift represent a block of time, and associate that block of time with a list of employees that are working then. This method is most intuitive: if Janaya and Melly are working Tuesday from 2 to 4, you would consider them to be working the same shift. However, this makes swapping and giving up shifts more difficult. If Melly wants to swap shifts with Harlin, Zhift must make sure it removes Melly from the shift (not Janaya), and puts Harlin in Melly's place.

Additionally, this concept does not make intuitive sense when considering overlapping shifts. If Janaya works 2 to 4 and Melly works 3 to 5, are Janaya and Melly working the same shift from 3 to 4? If so, now we need three shift objects to represent the 2 to 3, 3 to 4, and 4 to 5 shifts, and if not, our definition of a shift as having all the employees working then is validated.

A third issue occurs when somebody working a shared shift gives up their shift. If there is one employee per shift, it would be easy to think of the shift as unclaimed, but when there are multiple, the shift becomes…partially unclaimed? That confusion negates the advantages. Thus, a single shift instance should not have multiple people associated with it.

- <u>Allow shifts to have no employees</u>
  A shift with no employees might be desirable when a manager wants to create a shift, but hasn't decided who will work that shift yet. Allowing managers to create shifts without employees greatly complicates our design because of our concept of responsibility: if no one turns up for a shift, whoever was responsible for the shift is at fault. If a shift doesn't have an employee, nobody is responsible for that shift. This also complicates our design because it introduces a notion of claiming shifts that does not involve taking over the responsibility of a shift from someone else, which would require extending the set of records Zhift keeps, the email notifications Zhift sends, and the states of shifts in the database.

- <u>Allow shifts to have exactly one employee</u>
  Having a one-to-one correspondence between shifts and employees is the most elegant solution. No functionality is lost from Option 1, but changing shifts is much easier. The functionality of creating a shift without assigning an employee to it discussed in Option 2 is lost, but this actually makes it easier to fulfill Zhift's purpose of avoiding having shifts go unfilled, because Zhift is now enforcing the idea that managers should not create shifts if they do not have someone available to fill them.

How Should Swapping Work?

- <u>An employee who wants to give up their shift finds a different shift they can work and requests that shift</u>

Suppose Liza wants to give up a shift, and notices she can work Kai's shift. She requests Kai swap with her, but Kai doesn't get back to her for a couple of days, and then rejects the swap. Liza then has very little time left to find someone to take her shift, which might not be possible. This situation is undesirable and unavoidable.

- <u>An employee puts a shift up for swap, and other employees who can work that shift offer their shifts in return</u>
  Instead, Liza will put up her shift for swap and wait for other employees to offer shifts to her. Now, she gets a response as soon as the first of her fellow employees gets back to her, instead of waiting for her fellow employees one at a time. This makes it much easier for Liza to make sure her shift gets filled.

Should Swapping And Offering Be Mutually Exclusive?

- <u>No, employees can offer up a shift for claim and for swap at the same time</u>
  This option gives an employee the most chances of getting a shift filled, because the shift offered will appeal to fellow employees who are interested in picking up more hours and employees who would only want to swap. However, this muddles the differentiation between offers and swaps: offers are for when employees absolutely cannot work the shift, and swaps are for when employees still want to work that week, just at a different time.

- <u>Yes, employees can choose one or the other, but not both</u>
  This option keeps the distinction between offers and swaps very clear, and forces employees to be specific about what they want. If an employee offers up a shift for a swap and no one takes it, they can always change the offer type.

Should Zhift Support Shift Fragmentation?

- <u>Yes</u>
  Shift fragmentation when giving up shifts is a desirable feature. Suppose an employee is scheduled for 9 to 5 next Monday, but can't make their shift. If they can't find someone to cover for them for the whole day, they could split up their shift into 9 to 1 and 1 to 4, and get two different people to cover those time periods. Alternatively, maybe they can make 9 to 4, but have to leave early. Shift fragmentation would enable them to give up just the last hour of their shift.

  This is an ideal user experience, but is too complicated to implement in a five-week project. Fragmenting a shift would require getting rid of the original shift and creating two new ones for that week, and would result in a schedule that no longer followed the template schedule. This introduces an ugly tangle of complexities.

- <u>No</u>

Not allowing shift fragmentation keeps Zhift within a reasonable scope for a five-week project. There is a workaround to avoid the 9 to 5 problem discussed in Option 1: managers can define shifts more granularly, such as assigning the employee to eight consecutive one-hour shifts on Monday instead of a single eight-hour shift. Furthermore, since Zhift is targeting organizations where shift-changing occurs primarily between part-time employees, having single shifts of that length in the first place is an unlikely scenario.

Should Zhift Update Without Refresh?

- No
  Users expect the page to update as a result of their actions. Never updating the page without refresh is unreasonable for a modern web application.

- Yes, always
  Updating a logged in user's page as the result of another user's actions requires using sockets or some other form of server-client communication to notify the client of the change. This is by no means trivial to implement, and having two users logged in at the same time and interacting with each other's accounts is not a focus of this project.

- Yes, only as a consequence of the current user's actions
  This has the best of both worlds: the real-time responsivity expected without the complexity of the second approach.

Data Design Challenges

Should Manager Approval Be Required To Change Shift Responsibility?

- It should be an option
  Requiring a manager to approve shift changes (swaps and claimed offers) would help managers enforce constraints that are not easily programmed into a system, such as Anu and Xavier can't work together because they goof off too much or Sivan is training Lea so Lea can't work without Sivan. However, requiring manager approval complicates the data model's implementation because it adds an extra state for a change: once a change is ready to be made, there is now another state where it is pending manager approval, and another state if the manager rejects the change, for both swaps and offers. Keeping track of all that requires even more objects or properties, and complicates our already complicated data model.

- No
  While it would be a nice feature to have, it's not feasible for the scope of this project. Managers will still get notified about shift changes, and if need be they can manually

reassign the shifts or tell employees they're not allowed to work together. This keeps our data model simpler by limiting the number of states involved in a shift change.

Should Everything Be Embedded In An Organization?

- Yes
  Organizations never interact with other organizations, so keeping all the data associated with an organization embedded in the organization object greatly reduces the search space when searching for a specific object, because only that particular organization needs to be searched. However, searching in this way requires a lot more work, because instead of doing a query directly on the model of the object being searched for, we must instead find the organization and then search through its properties.

- No
  Embedding everything makes manipulating and even just finding data a nightmare. Mongo is optimized enough that doing queries that add an organization parameter isn't going to be slower than the embedded method. While testing, the size of the database won't be large enough that there would be any noticeable performance lags, and ideally if this were to scale, organizations would get their own database and there would no longer be a problem.

How Should Zhift Represent Shifts In The Database?

- Model: Shift
  The obvious solution is that Zhift has a single model, Shift, that represents a shift. However, this model is not expressive enough, because shifts can change ownership for different weeks. If an employee has a Monday night shift, and someone else covers it this week, next week it should still belong to the original employee. It would be possible to maintain a list of alternate employees responsible for shifts on the weeks they're covering, but that quickly becomes very complex to figure out who is working when. It also obscures the distinction between the default schedule that happens week to week, and a particular schedule for a particular week that may have some temporary changes in it.

- Models: Shift and TemplateShift
  Zhift could instead have two models that have to do with shifts: Shift and TemplateShift. The template shift is the shift that recurs every week, and the shift is a specific day that occurs once (see Concepts for more details, and the next design challenge for how these shifts are created). This makes the distinction between the default schedule and a particular schedule more transparent, and

makes determining who is responsible for a shift and changing the person responsible for a shift much easier, so this implementation will be used.

How Should Zhift Create New Shifts?

- As they are needed
  One option would be to only create shifts that employees or managers care about. When a user wanted to view or edit a shift, Zhift would check to see if that shift were in the database, and if not, would generate a new shift from the corresponding template shift. This would allow users to examine and change shifts at arbitrary points in the future, instead of limiting the shifts accessible to the next few weeks. However, the logic behind maintaining this system is quite complex, and the extra work on the backend when a user tries to access a shift that hasn't been created yet may slow down the user experience. This also exposes Zhift to attack: a malicious user could flood the database by writing a script to view shifts centuries into the future. Thus, this method is not practical.

- When someone logs in
  Instead, Zhift could create shifts for the next few weeks whenever a user logs in, avoiding the security vulnerabilities associated with creating shifts as needed. However, this still involves a lot of complicated work on the backend that could slow down the user experience when someone first logs in, especially if no one else has been on the system in a while.

- Automatically at regular intervals
  The cleanest way to approach this problem is to use a cron job to automatically generate the next few weeks of shifts at a regular interval. This is simple, and runs asynchronously to any user processes so it would not slow down the user experience.

What Happens When A Template Shift Changes?

- All shifts generated from that template shift change
  In this scenario, there a few issues to consider. What happens to shifts that have been swapped if the responsible person for a template shift changes? Is the swap still in effect? Does the new person in charge of that shift get it? The many edge cases involved in changing template shifts make this quite complex.

- Only future shifts generated from that template shift change
  One way to alleviate this complexity is to only apply the changes to the template shifts to new shifts that are generated, so all shifts that have already been

generated at the time of the change are unaffected. However, this approach requires the users to wait however many weeks shifts are generated in advance to see the changes, which is not desirable behavior.

- Make template shifts immutable
  If a manager really wants to change a template shift, they have to delete it (which deletes all shifts generated from it) and create a new one. This avoids all of the complexity and results in the most reasonable behavior.

Teamwork Plan

See separate teamwork plan document.