

ManageHelp

Design Document

Team 24

Tom Carsello

Matt Hiatt

Sharan Sivakumar

Jon Tokad

Table of Contents

Purpose	2
• Motivation	
• Functional Requirements	
• Non-Functional Requirements	
Design Outline	6
• High Level Overview	
• Sequence of Events	
• Data Flow Description	
• Activity Description	
Design Issues	9
• Functional Issues	
• Non-Functional Issues	
Design Details	12
• Description of Classes	
• UI Mockup	

Purpose

Motivation

A common issue that employees such as ourselves face at hourly jobs is a lack of efficient, convenient, and transparent scheduling applications. Workplaces around the country are still posting schedules on whiteboards, printing off excel tables, or telling employees verbally when they are supposed to work next. Additionally, time off requests are often made verbally or in handwriting which makes it difficult for managers to keep track of them and difficult for employees to trust that they will be honored. This makes it very difficult for employees to plan their weeks out and increases stress.

To mitigate these issues, ManageHelp provides a platform for managers to create, edit, and post schedules and for employees to view their weekly schedule, request days off, and trade shifts. In most workplaces, these functions are scattered across several platforms, are too cluttered and difficult to use, or not available at all. ManageHelp's objective would be to finally unify these functions. Having all these key workplace operations in one piece of software that is accessible from employee's personal computers means less time wasted going into the physical store to view schedules or repeatedly following off on time requests. It also means managers can focus on what is important in the store, serving customers and delivering a high quality product, instead of trying to manage folders of notes of time restrictions and other scheduling information. This would help reduce the high amount of stress employers and employees alike are under in service, fast food, and other industries.

Functional Requirements

- *Account and Workspace Management*
 1. As a user, I would like to be able to create my account on ManageHelp
 2. As a user, I would like to be able to login to my ManageHelp account
 3. As an admin, I would like to be able to create my company's ManageHelp workspace
 4. As an employee, I would like to be able to join my company's ManageHelp workspace
 5. As an admin, I would like to be able to remove and invite employees to our workspace
 6. As a user, I would like to be able to recover my password via email or security questions
 7. As an admin, I would like to be able to promote existing employees to managers
 8. As an admin, I would like to be able to assign tasks only managers can see

9. As an user, I would like to view my earnings with and without tax
- *Basic Managerial Functionality*
 1. As a manager, I would like to be able to assign roles and pay rates to employees
 2. As a manager, I would like to be able to view information about my employees in an easy to read format
 3. As a manager, I would like to be able to view expected labor costs for a day or week
 4. As a manager, I would like a way to post announcements
- *Manger Scheduling*
 1. As a manger, I would like to be able to assign shifts to employees
 2. As a manager, I would like to be able to input my company's weekly hours into the schedule template
 3. As a manager, I would like to be able to create weekly schedules several weeks in advance
 4. As a manager, I would like to be able to release and un-release schedules I am working on
 5. As a manager, I would like to be able to delegate scheduling permissions to shift managers
 6. As a manager, I would like to be able to receive overtime warnings if an employee is overscheduled
 7. As a manager, I would like to be able to attach specific tasks to an employee's shift
- *Basic Employee Functionality*
 1. As an employee, I would like to be able to view upcoming schedules when released by the manager
 2. As an employee, I would like to be able to go back and view previous schedules up to a month ago
 3. As an employee, I would like to be able to receive notifications when my schedule is updated
 4. As an employee, I would like to be able to input scheduling restrictions
 5. As an employee, I would like to see the specific tasks that I've been assigned to do by my manager
 6. As an employee, I would like to be able to check off any tasks I've completed during a shift
 7. As an employee, I would like a way to view and be notified of announcements
- *Coworker Shift Requests*
 1. As a manager, I would like to be able to veto employee shift trades/coverages
 2. As an employee, I would like to be able to request other employees cover a shift and accept other employees covered shifts
 3. As an employee, I would like to submit requests for certain days off

4. As a manager, I would like to be able to approve and reject employee day off requests
- *Ease of Access*
 1. As a user, I would like to be able to access ManageHelp on both desktop and mobile platforms
 2. As a user, I would like a way to easily navigate between different workspaces if I work two jobs
- *Chat Feature*
 1. As a user, I would like to have a system where I can communicate with other users via email or a chat system conveniently
 2. As a user, I would like the option to connect the chat feature to my email
 3. As a user, I would like the option to be notified when I receive a new chat

Non-Functional Requirements

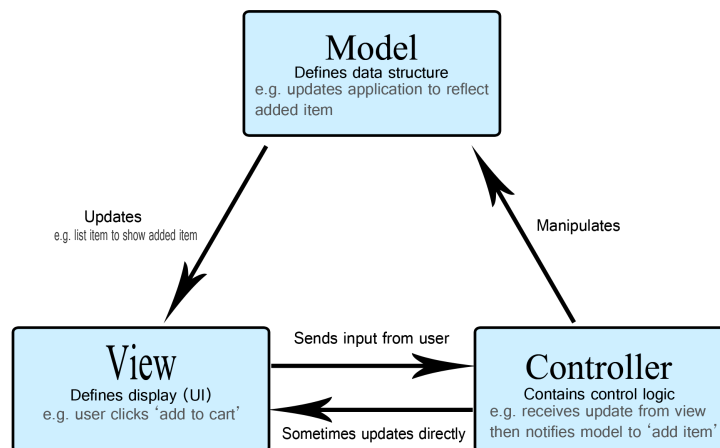
- *Architecture and Performance*
 1. The database shall be hosted through MongoDB Atlas's free AWS Cloud option
 2. Data storage will be supported up to 5GB of user data
 3. Data backed up with daily snapshots
 4. The application should be able to run without crashes
 5. The UI should launch within 5 seconds of launch
 6. The application should support at least 500-700 concurrent users (based on 1gb RAM from AWS Free Tier and roughly 50mb per concurrent)
- *Usability*
 1. The manager and employee UI should be similar aside from the differences in available functionality
 2. The UI needs to be intuitive and easy to learn to the average user, with no large prerequisite knowledge required
 3. Schedule creation should be fast and simple as managers often have more pressing issues when they are on the clock
- *Security*
 1. MongoDB Atlas provides crucial end to end database encryption
 2. MongoDB Atlas also provides the necessary authentication processes for retrieval of data by the controller
 3. Passwords must not be stored in plaintext
 4. User input must be cleaned to prevent webform hacking attacks
 5. Users should not be able to access employee information without the correct (managerial) access permissions

- *Scalability*
 1. The application should be constructed to allow portability between architectures
 2. The application's interface should scale properly when accessed from mobile devices
 3. The application should be able to handle spikes in usage or documentation by utilizing the automatic expansion of RAM memory on the shared AWS Cloud Server

Design Outline

High Level Overview

ManageHelp will be a web application that allows workplace users to post and view schedules, request time off, view hours worked and more. This project will use the Model-View-Controller (MVC) architectural pattern. The MVC pattern fits very well with the MERN stack since the React frontend forms a convenient representation of the “view” component of the MVC pattern. The middleware formed by Node.js and its framework Express.js forms the primary connection and handler between the frontend of React and the backend built as a MongoDB NoSQL database, hence making it our “controller”. Clearly this leaves the MongoDB database as the “model” since it is the database, so it defines the data structure and updates the user data when someone changes it in via the view -> controller pipeline. One slight adaptation of our pattern versus the classic MVC is that the controller will often update the view more than the model as it will be the one primarily manipulating and communicating with the structured data.



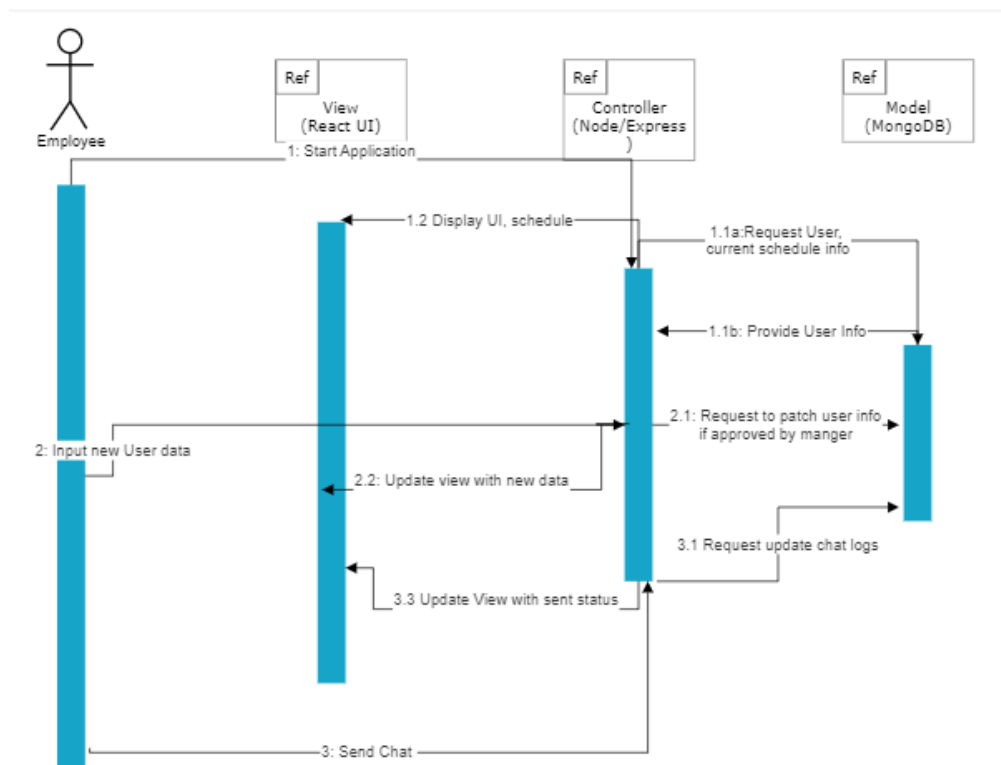
<https://developer.mozilla.org/en-US/docs/Glossary/MVC>

1. View
 - a. Provides the interface for the user
 - b. Updates interface when new data is sent through the model-controller pipeline
 - c. Lets controller know when users have entered new information, and that new information must permeate through to the model
2. Controller
 - a. Tells model when data needs to be updated
 - b. Makes requests (get, post, delete, etc) on behalf of the system
3. Model

- a. NoSQL database storing all user, organization, and schedule data
- b. Responds to updates my the controller
- c. Provides information to the view so the UI can be updated (when asked)

Sequence of Events Overview and UML Diagram

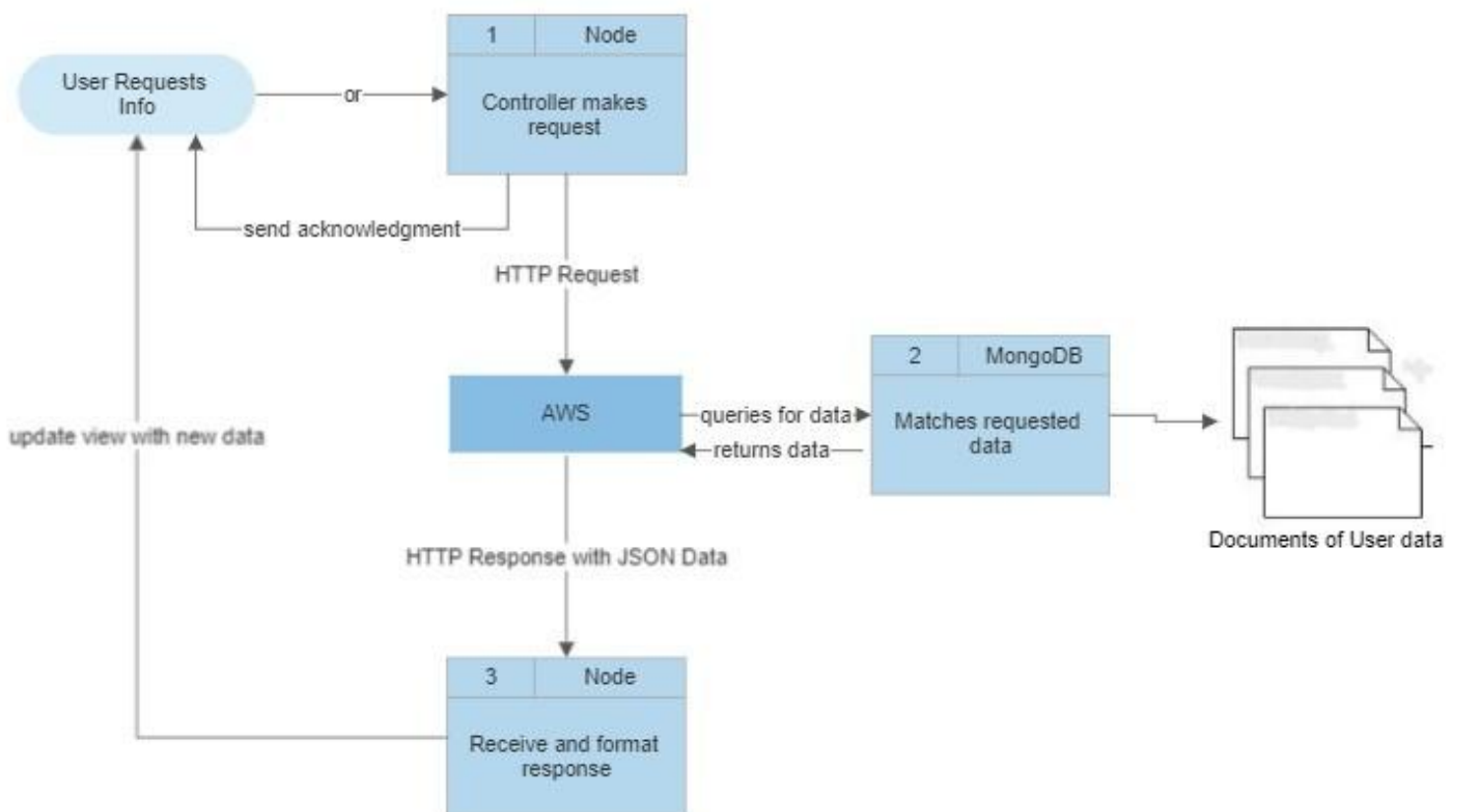
Our application event chain obviously starts with a user starting the application. When the user logs in, the controller notifies the model that authentication is necessary, which is supported within MongoDB. The user is authenticated and their view is updated to reflect their appropriate interface (manager or employee.) When the user interacts with the view, by clicking, entering text input, etc, the controller is always notified and either updates the server directly or makes the necessary queries to the model in the event data must be manipulated. If data is manipulated the model will respond to the controller and the view will then be updated. For example, an employee uses the view to see the chat page and enter input, when they click send the controller takes over and tries to send the message through to the server. The controller tells the model to update the chat received logs for the recipient and the chat sent logs for the sender, and notifies the controller of its success. The controller updates the view to reflect the message was sent successfully and the operation is done. The UML sequence diagram that follows exemplifies the interactions given that user data includes scheduling information, days off, etc. [Created with SmartDraw]



Data Flow Description

The user in the browser and the AWS hosted server will communicate through HTTP requests like is standard in any web application. So when the user makes a request to update, view, or remove some data in the view component, the controller will process that into an HTTP GET, PATCH, POST, etc request and relay it to the server. Then we can communicate with the NoSQL database at MongoDB who will search through “collections” which are groupings of individual user data stored in “documents” for the correct information and return. Then the server can return the data in JSON format to the controller and the controller can relay the information so the view can be updated to reflect the user’s request. The process is diagrammed below:

Data Flow Diagram



Design Issues

Functional Issues

1. What information do we require of General Managers to create a workspace?

- **Option 1: Email, name, password, workplace name**
- Option 2: Email, name, password
- Option 3: Name, password

It is clear that username and password are bare minimum requirements for registration of a workspace, otherwise the user would have no way to log into a unique account.

Requiring an email is also smart because it allows us flexibility in how we handle password resets and lost accounts, as well as a minimum of one method of sending notifications. However, we also want each ManageHelp workspace to have a one-one correspondence with a real workplace. This helps prevent duplicate workspaces and provides critical information to potential employees being invited to the space. We could allow this field to be added later, but we really can't allow the addition of employees until there is a name anyway so we might as well require it from the beginning hence we choose option 1.

2. How will workspace owners (General Managers) add employees to the workspace?

- Option 1: Require manager to preload an employee's information and share an invitation to the employee that, when accepted, will enter them into the workspace with an already set account.
- Option 2: Require manager to preload an employee's information and share an invitation to the employee that, when accepted, will enter them into the workspace with an already set account once they create a password.
- **Option 3: Have workspace be joinable to anyone with an organization code who signs up for an employee account with name and password, and let the manager edit their info upon them joining.**

Allowing employees to join the workspace with just an organization code allows for ease of joining and lets the user fill in their information themselves to avoid errors. It also allows us to not have to send invites through 3rd party paths and contain all the setup functionality within the web application. Option 2 also makes it difficult to associate one account with several workplaces if an employee had several jobs. Option 1 has the same issue but with the added undesirable of the manager knowing the employee's password.

3. How should a week's schedule be displayed in the GUI?

- **Option 1: As 7 rectangular columns, one for each day in the week, with employee shifts listed as blocks within each day, color coded according to a role or permission level, and sitting at a height corresponding to a time.**

- Option 2: As a list of employees working that day, with text next their name showing their startHour - endHour

Clearly we want to go with option 2, showing the schedule in a way that resembles most calendar apps will help make the UI easy to learn as most users may have some experience. A graphical element will also make it easier to see who is on shift simultaneously so optimal crews can be planned. Showing the schedule in mostly text would be easier and faster but is unintuitive and unappealing to the eye.

4. How should users be able to communicate about potential shift coverages/trades?
 - Option 1: Making formal shift for shift requests available for all other equally or higher permissioned employees to see.
 - Option 2: Installing a chat feature so that employees can work out a shift trade between themselves.
 - **Option 3: Both.**

We want both a way for employees to request shift trades and a way for them to talk. We need a mechanism for the employees to submit requests to the managers for approval of course, but if an employee had to keep sending out slightly different requests until they could get a coworker to be able to fill it that would be inefficient. It would be easiest if an employee could post an initial request, then interested coworkers could send chats to work out a good deal and the request could then be updated and sent for approval.

5. How do we track labor costs for a weekly schedule?
 - Option 1: Keep employee hours, overtime hours, pay rate, and overtime rate, values stored within an employee object and calculate pay by summing the products of hours * rate and overtime hours * overtime rate, then average costs over all the employees on a shift.
 - **Option 2: Do the same but do not track overtime hours, instead keep a workplace variable that lets us know how many hours constitute overtime work.**

We choose option 2 because it allows us to track a smaller amount of data per employee. It is easy enough to compute the necessary labor costs from constant equations when they need to be displayed or used. The overtime hours per employee is easily calculated with subtraction of their total hours and a constant number set at an organizational level.

Non-Functional Issues

1. What database vendor should we use?
 - **Option 1: MongoDB**
 - Option 2: MySQL
 - Option 3: Cloudera

- Option 4: Firebase

We choose option 1, MongoDB for our backend/database vendor since it is very popular for projects of all sizes. It also gives us an advantage over MySQL in that our data does not have to fit cleanly into tabular schema. We can choose to format user documents however we choose for convenience and speed. It also has superior documentation and support to smaller services like Cloudera or Firebase.

2. What server side backend framework or language best suits our needs?

- Option 1: Java
- **Option 2: Node.js**
- Option 3: Django

Although we all have familiarity with Java, we still are electing to go with Node.js since we are building a web application and Node is an extraordinarily popular option for such development at the moment. Its framework Express.js also allows easy use of RESTful API. While Django is another valid option, as a team we lack python familiarity and Django does not seem to be associated with MongoDB often.

3. What frontend should we use to round out our techstack

- Option 1: Angular
- Option 2: Bootstrap
- **Option 3: React**

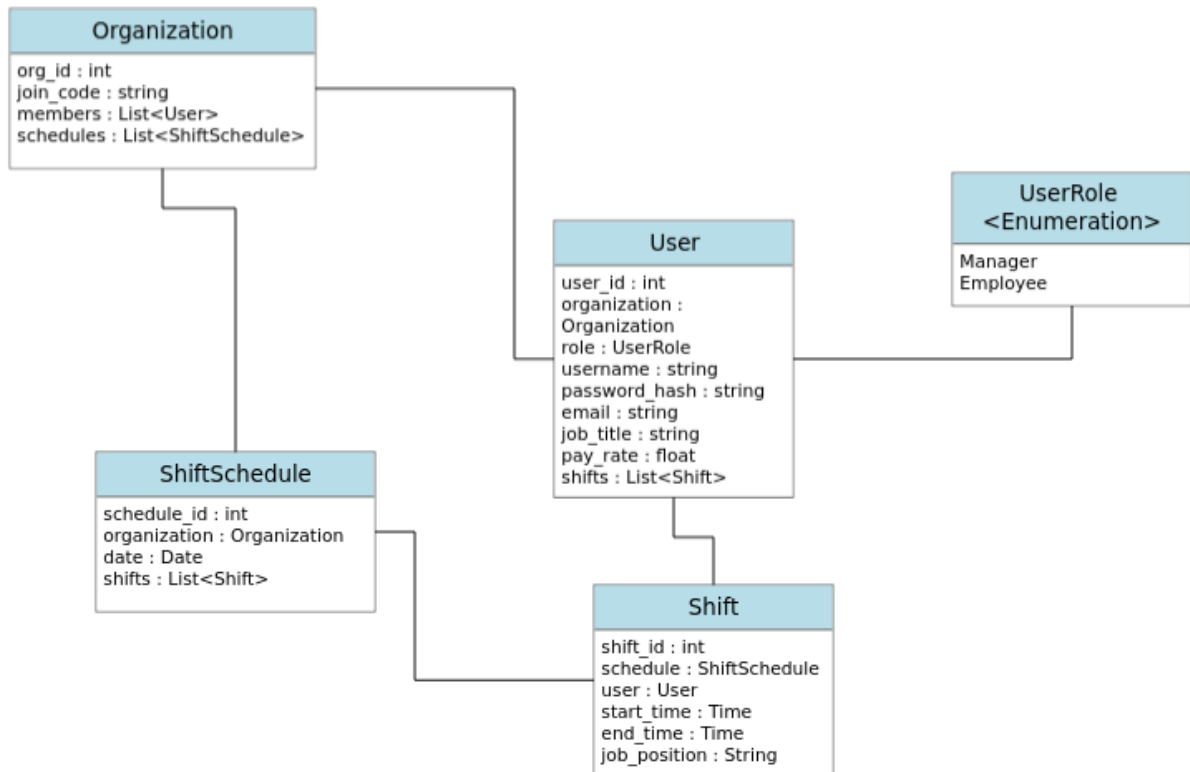
ReactJS is clearly the logical choice to round out our techstack, commonly referred to as MERN (MongoDB, Express, React, Node.) It works in seamless coordination with Node as well. There is also a plethora of documentation and tutorial materials online, and we have some slight exposure to React as a group already. Angular is another good choice but since we have less exposure we elected to stick with React. While Bootstrap is a popular framework in its own right, it seems to slot into the rest of our stack slightly worse than React or Angular would.

4. Where will we host our non-database backend?

- **Option 1: AWS**
- Option 2: Google Cloud
- Option 3: Azure

We choose AWS in large part because of our use of MongoDB Atlas for our database. Atlas offers hosting via AWS as its first option, and since all three of these options offer free tiers of hosting we elect to stay consistent through our project and use AWS to host our server.

Design Details



Description of Classes and their Interactions

Note: These classes have been designed based on the objects that we used in our application.

Each object has a list of attributes that represents the characteristics of said object.

Organization

- An Organization refers to a specific company using our platform.
- A new Organization is created when a company wishes to use our platform, and registers their organization
- Each Organization will contain a unique identification number
- Each Organization will contain a join code which will be entered by Users as they register, so that they may join their Organization.
- Each Organization will contain a list of ShiftSchedules, which is a list of all defined schedules for that Organization

User

- A User refers to a specific individual using our platform
- Each User will have a unique identification number
- Each User will contain a reference to the organization to which it belongs
- Each User will contain a UserRole to define its permissions and available functions
- Each User will contain a username, password hash, and email that are given by the User when they register
- Each User will have a job title and pay rate, each assigned to that User by a manager
- Each User will contain a list of Shifts in which that User is scheduled to work

UserRole

- A UserRole is an enumeration to define what type of functionality a User will have
- A UserRole is created alongside each User
- Employee is default
- Manager is privileged, and has the ability to access more secure features

ShiftSchedule

- A ShiftSchedule represents the schedule for a given day in the Organization
- A ShiftSchedule is created by the managers of an Organization and is viewable by all Users including employees.
- Can only be directly edited by managers
- Each ShiftSchedule contains a unique identification number
- Each ShiftSchedule contains a reference to the organization of which it has been created under
- Each ShiftSchedule will contain the date for which the ShiftSchedule pertains
- Each ShiftSchedule will contain a list of Shifts

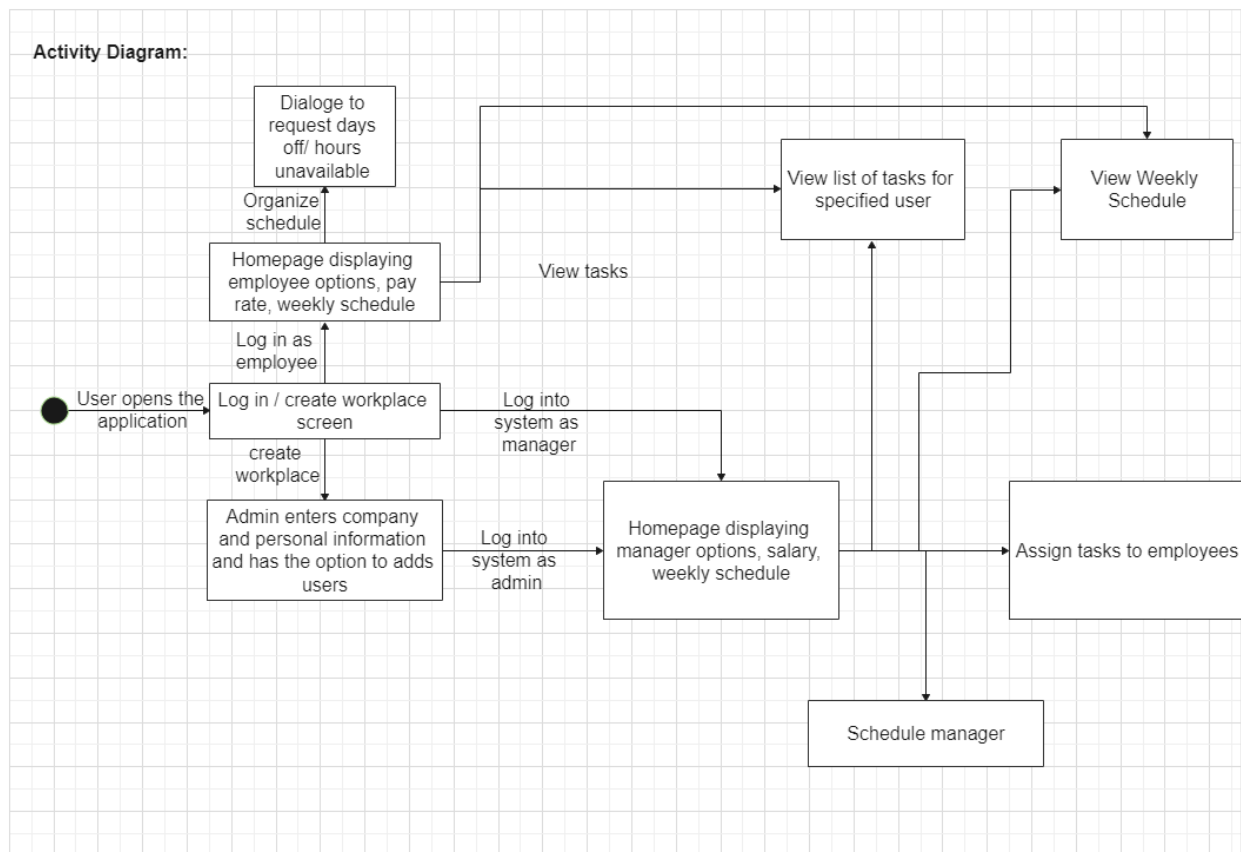
Shift

- A Shift represents a singular shift scheduled for a specific user.
- A Shift is created when a manager creates one for a ShiftSchedule
- A Shift contains a unique identification number
- A Shift contains a reference to the ShiftSchedule it belongs to
- A shift contains a reference to the User who is working the shift
- A shift contains the start and end times that define when the User should work
- A shift contains the name of the job the User is assigned to work for that shift (i.e. Cashier, Line cook, Supervisor, etc.)

Activity Description

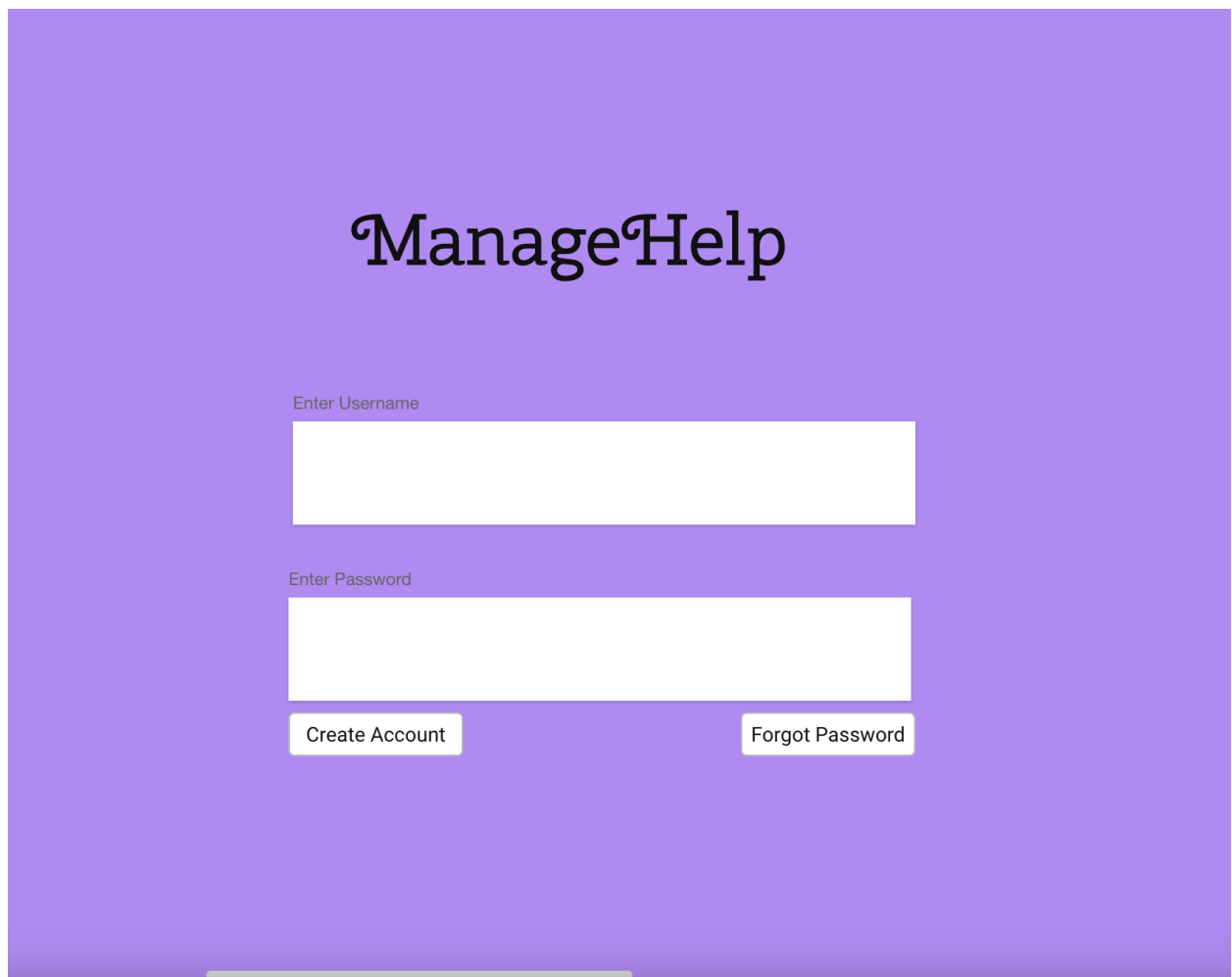
Since administrators and managers have the same permissions except for the initial setup, their homepages look the same, except that the admin has permission to assign managers tasks. Employees, however, need their own interface to do their own thing without interfering with the organization of the company. The admin/managers are able to handle organization concurrently. Only admins are allowed to create accounts for employees and managers. There is an option to get your password back if you have forgotten it.

Activity Diagram



UI Mockup

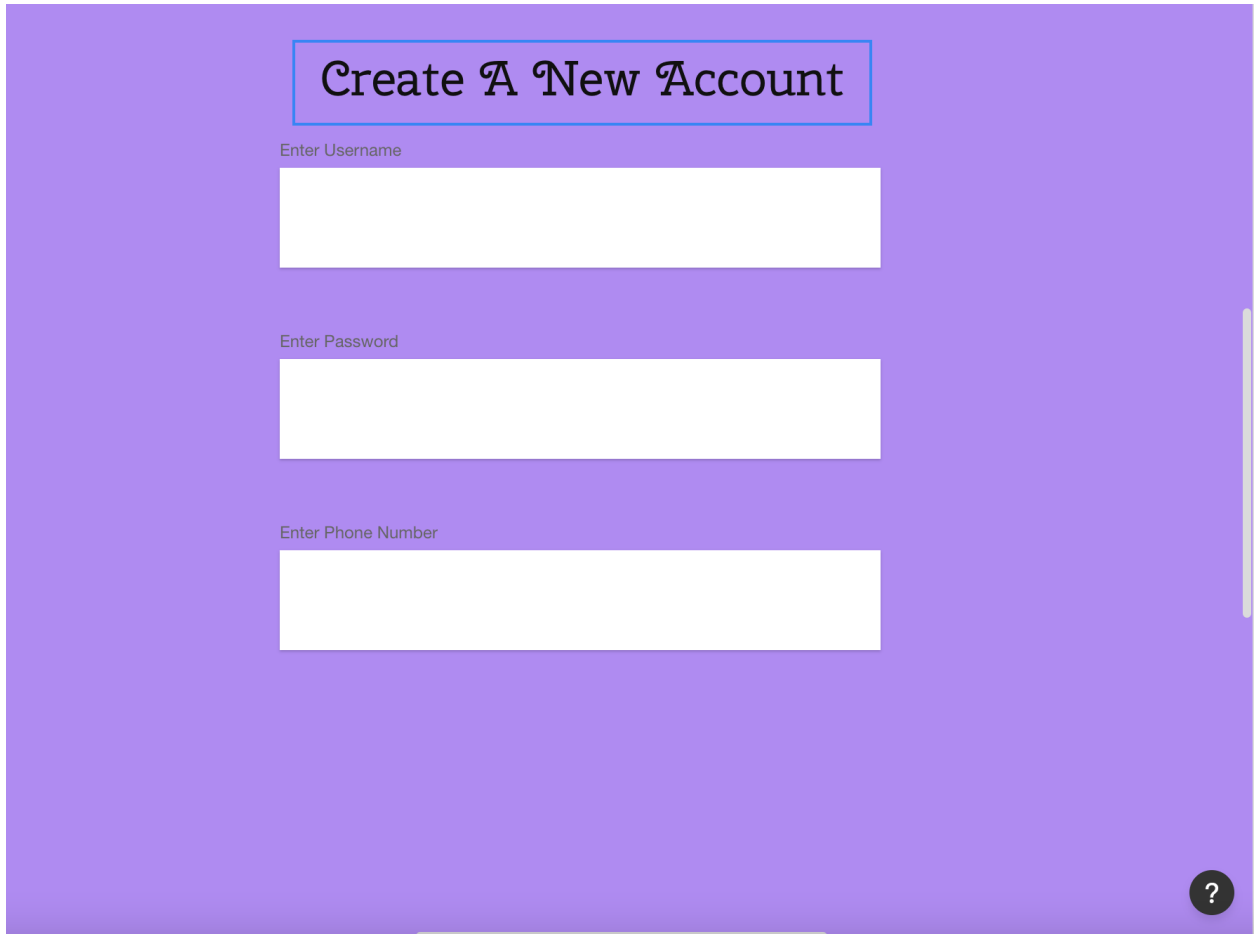
An important part of our web application is our user interface because it is how the user interacts with the underlying code within our web application. We wanted our UI to be user friendly, efficient, and simple to understand for ease of use. The mockups show what buttons and functions we will include, although in a much more elementary design/art method that we can create with our actual code.



The image shows a UI mockup for a login page titled "ManageHelp". The background is a solid purple color. At the top center, the text "ManageHelp" is displayed in a large, black, serif font. Below the title, there are two input fields. The first field is labeled "Enter Username" in a small, grey font above it. The second field is labeled "Enter Password" in a small, grey font above it. Below the password field, there are two buttons: "Create Account" on the left and "Forgot Password" on the right. Both buttons are white with black text. At the bottom of the page, there is a thin, horizontal, light grey line.

This is the home page. The home page has a simple user interface where the user has the option to either log in with an existing account, create a new account, or request a new

password by clicking “Forgot Password”. There are two textboxes to accept the string input that is the user’s username and password as well as two buttons to navigate to the page to create an account and the page to request a new password

A screenshot of a web application's sign-up page. The page has a solid purple background. At the top center, there is a white rectangular box with a thin blue border containing the text "Create A New Account" in a black serif font. Below this title, there are three white rectangular input fields stacked vertically. Each field is preceded by a small, light gray label: "Enter Username" for the first, "Enter Password" for the second, and "Enter Phone Number" for the third. In the bottom right corner of the purple area, there is a small black circle containing a white question mark. A thin, light gray vertical scrollbar is visible on the right edge of the page.

This is the sign up page where a user can choose to create a new account if they do not have one. This page contains three user input boxes where the user can input their username, password, and phone number in order to sign up for the web application.

User Information

Are you a manager or a

☒ Manager
☐ Employee

What is the name of your company?

Text Input

What is your name? (First and Last)

Text Input

What is your role in the company?

Text Input

?

This is the User Information page. This page is either accessed for the first time when a user creates an account or it can be accessed from the settings dropdown menu. This page contains several labels and input text boxes to ask the user questions about their account.

Worker View

Settings ▾

Input any scheduling restrictions here:

View Schedule

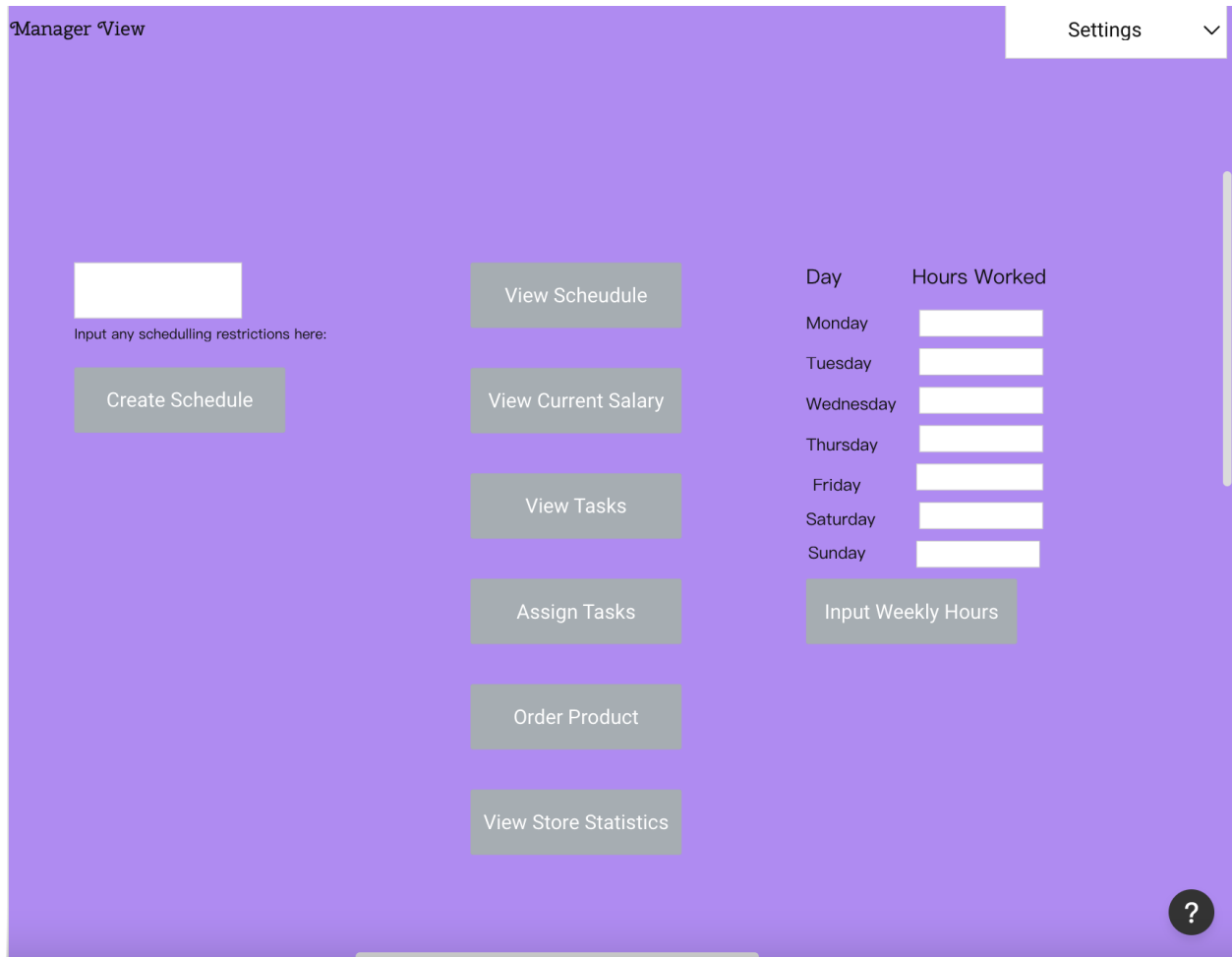
View Current Salary

View Tasks

Day	Hours Worked
Monday	<input type="text"/>
Tuesday	<input type="text"/>
Wednesday	<input type="text"/>
Thursday	<input type="text"/>
Friday	<input type="text"/>
Saturday	<input type="text"/>
Sunday	<input type="text"/>

?

This is the worker view page. The work view page contains essential information to the user if they are of the worker status. In the top right corner there is a settings dropdown menu that has three options: Change Username, Change Password, and Change Phone #. Directly below that we have several input boxes for the user to input the amount of hours they worked per day of the week. To the left of that, the user can view important information such as their schedule, their salary, tasks to do and more using buttons. The last button was left blank because this is just a mock-up and doesn't may be changed.



The image shows a mobile application interface for a "Manager View". The header is purple with the text "Manager View" on the left and a "Settings" button with a dropdown arrow on the right. The main content area is white and contains several interactive elements:

- A text input field with the placeholder "Input any scheduling restrictions here:" and a "Create Schedule" button below it.
- A vertical column of buttons: "View Scheudule", "View Current Salary", "View Tasks", "Assign Tasks", "Order Product", and "View Store Statistics".
- A table for tracking hours worked per day:

Day	Hours Worked
Monday	<input type="text"/>
Tuesday	<input type="text"/>
Wednesday	<input type="text"/>
Thursday	<input type="text"/>
Friday	<input type="text"/>
Saturday	<input type="text"/>
Sunday	<input type="text"/>

Below the table is an "Input Weekly Hours" button. A circular help icon with a question mark is located in the bottom right corner.

This is the manager view page. The manager view page has a lot of the same features as the worker view page such as the ability to input hours worked per day and the settings tab. The difference would be that the manager view page has additional functionality such as ordering product, assigning tasks, viewing store labor/other statistics, creating schedules, and more due to the fact that a manager would have more permissions than a regular employee.