Sequoia Grove

Weekly Employee Scheduling Application

Bethany Armitage, Ted Pascua, Amador Silva, Jasjot Sumal

CMPS 490B, Fall 2016, Melissa Danforth

Due Friday June 3rd, 2016

Table of Contents

Executive Summary

Sequoia Grove is a web application designed to simplify employee scheduling. It was designed for businesses following a scheduling system where work schedules with shifts are written and published weekly. The model for this product was constructed by researching the current system in place by a local Delicatessen, The Sequoia Sandwich Company. The application then expands on the model by being built generically enough to accommodate any business operating on a similar basis.

There are many advantages to programmatically accomplishing a task such as this. In general, such a task is laborious and error prone when done by hand because of the amount of consideration that goes into employee scheduling. The availability of each individual employee must be taken into account, as well as individual vacation requests. Shifts are worked only by an employee who is trained in the required position. Balancing minimum and maximum hours each employee is scheduled becomes tedious because data regarding the current schedule such as the number of hours or days each employee is scheduled for is only available when calculated by hand. Sometimes an employee does not work well with another employee, and would be best scheduled without overlapping shifts. When scheduling by hand sometimes employees are accidentally “double scheduled” or scheduled twice on a day, where it would be physically impossible for them to work both shifts at the same time.

Sequoia Grove features a user friendly real world design for managing employees and schedules. User permissions are in place to fine tune which features each individual user may access. At the very minimum, a user can view any published schedules, and any features after that require the account holder to grant permissions for access. By default employees may view the schedule and submit requests for vacation. Managers additionally can manage vacation requests, edit employee information, and edit the schedule. Account holders can additionally manage store information such as holidays, shifts, and expected weekly deliveries.

Since this project was a continuation from databases class project, it started out with some functionality.

At the time the project begun, two user groups were in place for accessing the front end website: employees and managers. Managers will need to make heavier use of the front end on a daily basis due to their daily responsibilities, and correspondingly are provided more functionality and access available to them.

Once any user logged in, they were taken to the site's home page, which shows weekly schedules that have been finalized by managers. The focus of this website is to provide a centralized point for modifying and viewing work schedules. Employees are only expected to check the schedule on a daily basis to determine their hours for the week, since they will not be modifying the schedules. The employee user group was therefore initially only given access to the site's home page, however this was changed as the website was expanded during the senior project.

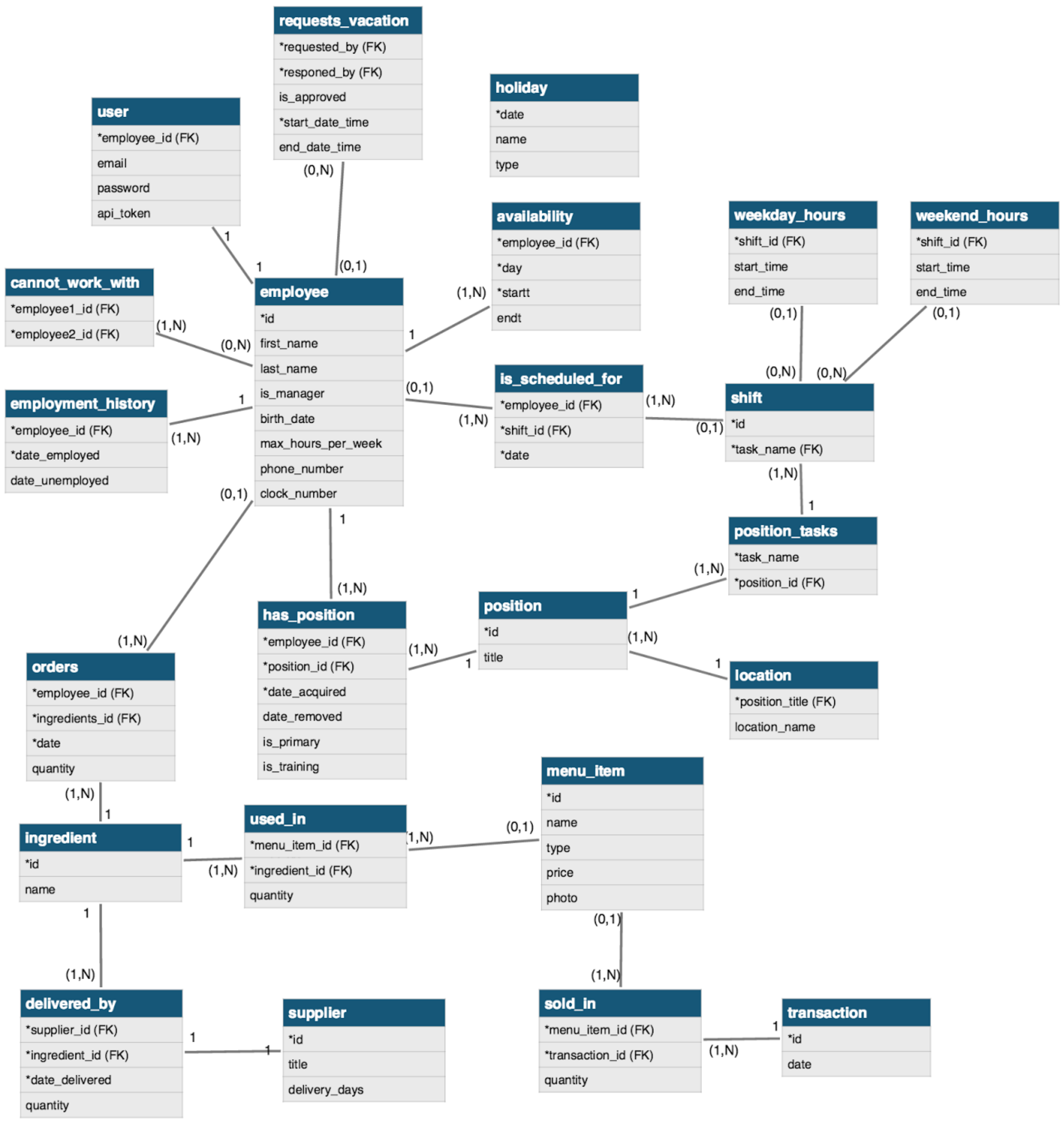
Managers on the other hand need to be able to create and modify work schedules, in addition to viewing the schedule, and are therefore given access to additional web pages. Upon login, managers were taken to the home page, like employees, where they may view schedules.

In addition, however, managers may visit the "Employee" tab where they can search for any employee, active or inactive. This tab allows managers to view and modify basic information about the employee such as the employee's name and birthday, as well as scheduling information such as weekly availabilities as positions worked by the employee. The employee's employment history with the company is also listed on this tab. Unlike the other modifiable pieces of information on this tab, employee availability is likely to change by the week, so it is expected managers will be visiting this tab often to adjust this information in preparation for creating next week's schedule.

The "Scheduling" tab is also available to managers, where they may adjust which employees are scheduled for the shifts in a given week. It is expected managers will visit this page the most while accessing the site, most likely on a daily basis, in order to create and prepare the following week's schedule. Correspondingly, most of the website's functionality is focused on this specific webpage, in order a facilitate quickly and easily scheduling employees for shifts. For example, to quickly populate an upcoming week's schedule, managers may import the schedule from the previous week if they know the following week's schedule is going to be very similar. This reduces the amount of shifts that need to explicitly scheduled down to the few that need to be changed.

Finally, a “Data” tab was provided for managers to access, although it has since been removed as the essential information on this page is now available in the Scheduling tab. This page checked data on the currently active schedule to calculate the totals number days and hours each employee works on that given week. The information was presented in graphical format using charts and graphs, allowing managers to easily visualize how much work employees are being given in comparison to each other. Manager were expected to visit this page on a weekly basis before finalizing the following week’s schedule in order to compare how much work employees are being given.

A database is needed to store all the information being presented and handled by the front end, and this was developed with the initial project as well. The original Relational Diagram of the database is given as follows.


requests_vacation
*requested_by (FK)
*responed_by (FK)
is_approved
*start_date_time
end_date_time
(0,N)
(0,1)
holiday
*date
name
type
user
*employee_id (FK)
email
password
api_token
1
1
availability
*employee_id (FK)
*day
*startt
endt
(1,N)
1
weekday_hours
*shift_id (FK)
start_time
end_time
(0,1)
weekend_hours
*shift_id (FK)
start_time
end_time
(0,1)
cannot_work_with
*employee1_id (FK)
*employee2_id (FK)
(1,N)
(0,N)
employee
*id
first_name
last_name
is_manager
birth_date
max_hours_per_week
phone_number
clock_number
is_scheduled_for
*employee_id (FK)
*shift_id (FK)
*date
(0,1)
(1,N)
(1,N)
(0,1)
(0,N)
(0,N)
shift
*id
*task_name (FK)
employment_history
*employee_id (FK)
*date_employed
date_unemployed
1
(1,N)
(0,1)
1
(1,N)
1
position_tasks
*task_name
*position_id (FK)
(1,N)
1
(1,N)
has_position
*employee_id (FK)
*position_id (FK)
*date_acquired
date_removed
is_primary
is_training
(1,N)
1
position
*id
title
(1,N)
1
location
*position_title (FK)
location_name
(1,N)
orders
*employee_id (FK)
*ingredients_id (FK)
*date
quantity
(1,N)
1
ingredient
*id
name
1
(1,N)
used_in
*menu_item_id (FK)
*ingredient_id (FK)
quantity
(1,N)
(0,1)
menu_item
*id
name
type
price
photo
(0,1)
(1,N)
1
(1,N)
delivered_by
*supplier_id (FK)
*ingredient_id (FK)
*date_delivered
quantity
1
1
supplier
*id
title
delivery_days
sold_in
*menu_item_id (FK)
*transaction_id (FK)
quantity
(1,N)
1
transaction
*id
date

Summary of Initial Relational Diagram

(Relations kept throughout Senior Seminar)

`Availability` - stores data about hours that current employees may be scheduled for on specific weekdays.
`Cannot Work With` - stores which employees an employee has difficulty working with
`Employment History` - stores employment history of current and previous employees.
`Has Position` - associates an employee with a position
`Holiday` - stores information about dates the store is closed for a full or half day.
`Ingredient` - stores data regarding ingredients used to create food items
`Is Scheduled For` - associates an employee with a shift and date
`Position` - stores data about various positions that employees occupy.
`Shift` - defines the expected tasks that need to be scheduled.
`User` - stores employee login information. This is a subclass of the employee entity (later merged with Employee).

(Relations removed during Senior Seminar)

`Delivered By` - associates a delivery with an ingredient
`Employee` - stores data about current employees.
`Location` - stores data about a position's location in the store
`Menu Item` - stores data regarding food items that may be sold.
`Orders` - associates an ingredient with an employee and date
`Position Tasks` - stores the task names a position can take
`Sold In` - associates a menu item with a transaction
`Supplier` - describes which companies deliver ingredients to the company.
`Transaction` - stores sales transactions that have been made by the company.
`Used In` - associates an ingredient with a menu item
`Weekday Hours` - stores the start and end times for shifts that need to be scheduled for weekdays. This is a subclass of the shift entity.
`Weekend Hours` - stores the start and end times for shifts that need to be scheduled for weekends. This is a subclass of the shift entity.

As the project was developed during senior seminar, the database was expanded and modified to make queries more concise, and also allow for different user permissions and further login functionality. The front and back end have also been modified greatly; further pages and functionality have become available to both employees and managers. All these changes are covered later on in the paper.

Brief overview of the development process

The initial project goal had two tiers, with the first tier split up into three phases.

Tier 1 Phase 1 included tasks to setup development environments on each team member's machine, and start with tasks touching each part of the application in order to have each team member become acquainted with the entirety of the project. This phase focused on refining the project in its current state, cleaning up small issues, and preparing it for more major development in the remaining phases and tiers.

Tier 1 Phase 2 focused on adding security features. Other major features that were relatively simple to complete were also worked on during this phase. These included implementing holidays, the ability to employ/unemploy employees, and ensuring the schedule makes several checks to ensure employees are not scheduled during the wrong times.

Tier 1 Phase 3 was designed to overhaul development by implementing new development tools and changing the host servers for the database and web application. However, once this phase was reached, other issues were added on and became the focus of the phase due to time constraints.

Tier 2 held all remaining issues that need to be added to the project to make the user experience more smooth, or that would make the application much more functional. These were the features that were considered nice to have, and nonessential to the completion of the project. They included auto-generating the schedule based on prior weeks in history, improving the login, making the application mobile friendly, adding support for multiple locations, adding push notifications, and adding session logging.

One planned major change was the translation of the existing oracle database to postgresql. At first, the task did not appear to be such a major undertaking. The SQL for both oracle and postgresql were similar. One major difference was that postgresql would not support procedures. It supports functions that return void instead. This doesn't seem like a big difference, but the queries involving procedures would always throw an error because it had an unexpected return value. This was overcome by using Java to catch the SQLException and continue running as usual. At the same time the database was translated, liquibase was implemented for database versioning. The implementation was fairly straight forward. A plugin was added to Maven, the build tool, to check the liquibase changelogs upon application startup. A file called db.changelog.xml is used to list all changelogs. This file is read in, and any new additions are then parsed and run against the existing database. The database then needed to be described in terms of changelogs to run. The task of translation to postgresql and implementation of liquibase proved to take more time than expected. One small development error was that a hard-coded value was written in for testing, and the application would not reflect expected results. The error was assumed to be from the change of databases, or liquibase instead. This small error took a long time to solve. It was a major hangup for the task overall.

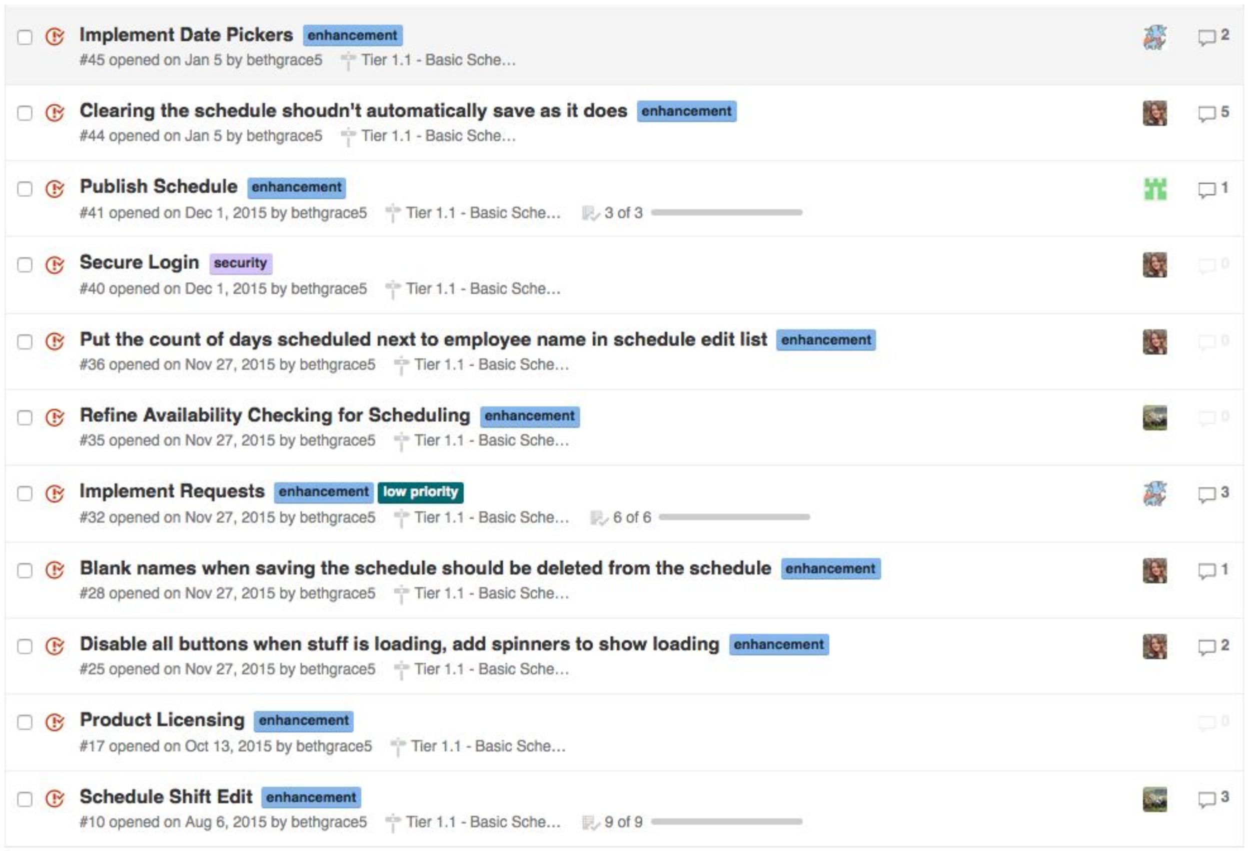
Another major feature to overcome was application security. Initially, Mozilla Persona was to be used as an authentication service, however, Mozilla decided to end support for the service. The next authentication service implemented was using authentication built into the system. This in house authentication was never fully developed. It extended to essentially sending an encrypted password to verify it in the database. This worked well, the problem, however, existed with the extent of additions in house authentication would require. It would have to handle cases where a user might have forgotten their password. It would have required a secure way to set an initial password for each user. It would also require checking that passwords set were secure. Some of these additions would not be able to be implemented without the addition of an email service. Finally, Google login service using the google api for oauth2 was implemented. This solved the above problems, it is secure, and reliable. Any security flaws are not likely to surface, but if they do, the blame would most likely be on Google instead of this application. The only caveat, however, is that all users must have a Google account to sign in with the application. Google accounts are free, so at most this becomes a minor annoyance to those users without a Google account.

Additionally, as the application was developed, best practices were learned. The front end originally relied on keeping states within Angular controllers. This is not best practice, because it led to repetition of code, and inefficiency because controllers are loaded and unloaded as they enter and leave the scope. Also, sharing data between controllers became a nightmare. What was learned was that Angular factories and services exist to keep a state and are easily shared within any controller they are injected as a dependency of. They are not unloaded and loaded as the scope changes, so they are more efficient. They also reduce code so that the controllers are less cluttered overall. Angular factories and services can be related to a class in an object oriented language. They have internal variables and functions, and exposed variables and functions. The addition of factories and services helped make the project more maintainable as well.

Generally, the development of the application followed the tiers set in place, but fell behind the planned schedule. To address this problem, the project issues were reprioritized so that issues in subsequent tiers were started before all the issues in prior tiers were completed. Issues that were not relevant to the completion of the senior seminar project, but potentially still important to project, were put aside. The issues put aside included migrating the database and to a new host server, moving the web application onto a host server as well, SSL certificates, improving efficiency in some areas, fixing some bugs, and fully completing the schedule checks. Those that were prioritized included API Token security, fixing the scheduling UI, employing/unemploying employees, installing Liquibase for developing on personal machines, implementing holidays, implementing an auto-generating schedule, and adding a new login service. The prioritized issues were picked based on what was needed most immediately, and, towards the end of the quarter, what could be completed most quickly. A breakdown of issues by tier is given below. Fully completed issues are shown with red icons while partially completed or unstarted issues are shown with green icons.

Summary of Tiers and Completed Issues

Summary of Tier 1.1



Implement Date Pickers enhancement
#45 opened on Jan 5 by bethgrace5 · Tier 1.1 - Basic Sche... · 2

Clearing the schedule shoudn't automatically save as it does enhancement
#44 opened on Jan 5 by bethgrace5 · Tier 1.1 - Basic Sche... · 5

Publish Schedule enhancement
#41 opened on Dec 1, 2015 by bethgrace5 · Tier 1.1 - Basic Sche... · 3 of 3 · 1

Secure Login security
#40 opened on Dec 1, 2015 by bethgrace5 · Tier 1.1 - Basic Sche... · 0

Put the count of days scheduled next to employee name in schedule edit list enhancement
#36 opened on Nov 27, 2015 by bethgrace5 · Tier 1.1 - Basic Sche... · 0

Refine Availability Checking for Scheduling enhancement
#35 opened on Nov 27, 2015 by bethgrace5 · Tier 1.1 - Basic Sche... · 0

Implement Requests enhancement low priority
#32 opened on Nov 27, 2015 by bethgrace5 · Tier 1.1 - Basic Sche... · 6 of 6 · 3

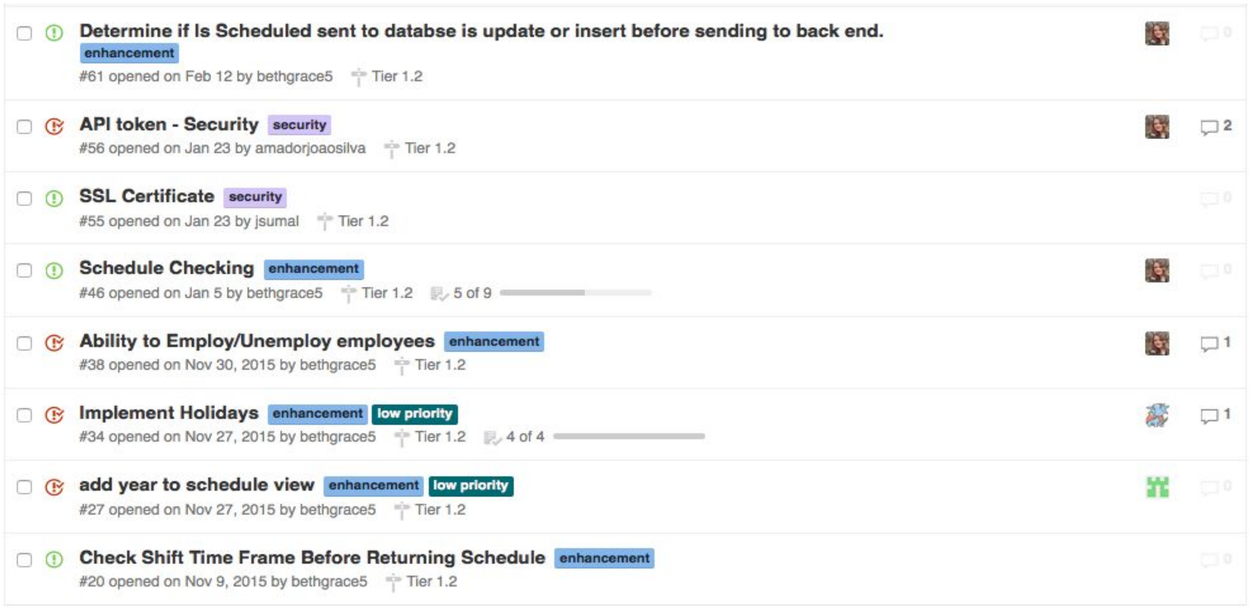
Blank names when saving the schedule should be deleted from the schedule enhancement
#28 opened on Nov 27, 2015 by bethgrace5 · Tier 1.1 - Basic Sche... · 1

Disable all buttons when stuff is loading, add spinners to show loading enhancement
#25 opened on Nov 27, 2015 by bethgrace5 · Tier 1.1 - Basic Sche... · 2

Product Licensing enhancement
#17 opened on Oct 13, 2015 by bethgrace5 · Tier 1.1 - Basic Sche... · 0

Schedule Shift Edit enhancement
#10 opened on Aug 6, 2015 by bethgrace5 · Tier 1.1 - Basic Sche... · 9 of 9 · 3

Summary of Tier 1.2


Determine if Is Scheduled sent to databse is update or insert before sending to back end. enhancement
#61 opened on Feb 12 by bethgrace5 Tier 1.2 0

API token - Security security
#56 opened on Jan 23 by amadorjoaosilva Tier 1.2 2

SSL Certificate security
#55 opened on Jan 23 by jsumal Tier 1.2 0

Schedule Checking enhancement
#46 opened on Jan 5 by bethgrace5 Tier 1.2 5 of 9 0

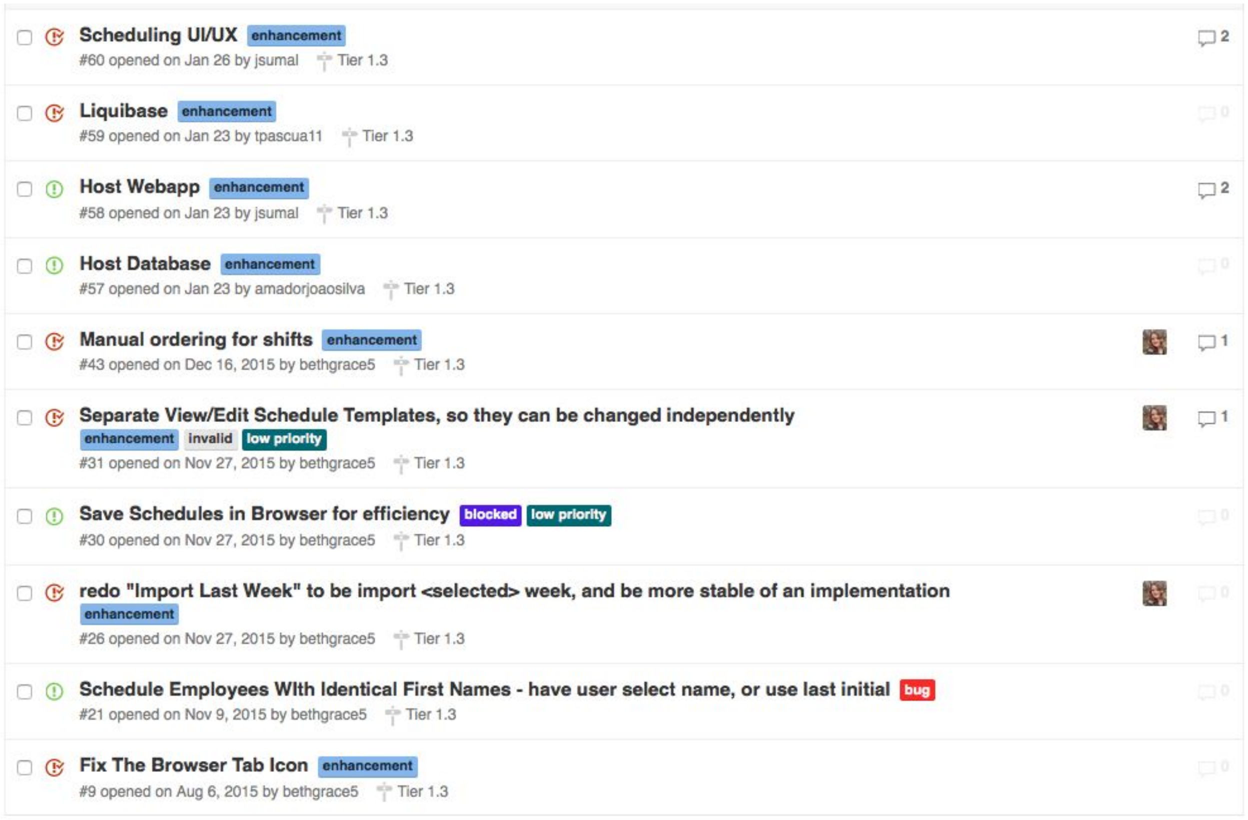
Ability to Employ/Unemploy employees enhancement
#38 opened on Nov 30, 2015 by bethgrace5 Tier 1.2 1

Implement Holidays enhancement low priority
#34 opened on Nov 27, 2015 by bethgrace5 Tier 1.2 4 of 4 1

add year to schedule view enhancement low priority
#27 opened on Nov 27, 2015 by bethgrace5 Tier 1.2 0

Check Shift Time Frame Before Returning Schedule enhancement
#20 opened on Nov 9, 2015 by bethgrace5 Tier 1.2 0

Summary of Tier 1.3


Scheduling UI/UX enhancement
#60 opened on Jan 26 by jsumal Tier 1.3 2

Liquibase enhancement
#59 opened on Jan 23 by tpascua11 Tier 1.3 0

Host Webapp enhancement
#58 opened on Jan 23 by jsumal Tier 1.3 2

Host Database enhancement
#57 opened on Jan 23 by amadorjoaosilva Tier 1.3 0

Manual ordering for shifts enhancement
#43 opened on Dec 16, 2015 by bethgrace5 Tier 1.3 1

Separate View/Edit Schedule Templates, so they can be changed independently enhancement invalid low priority
#31 opened on Nov 27, 2015 by bethgrace5 Tier 1.3 1

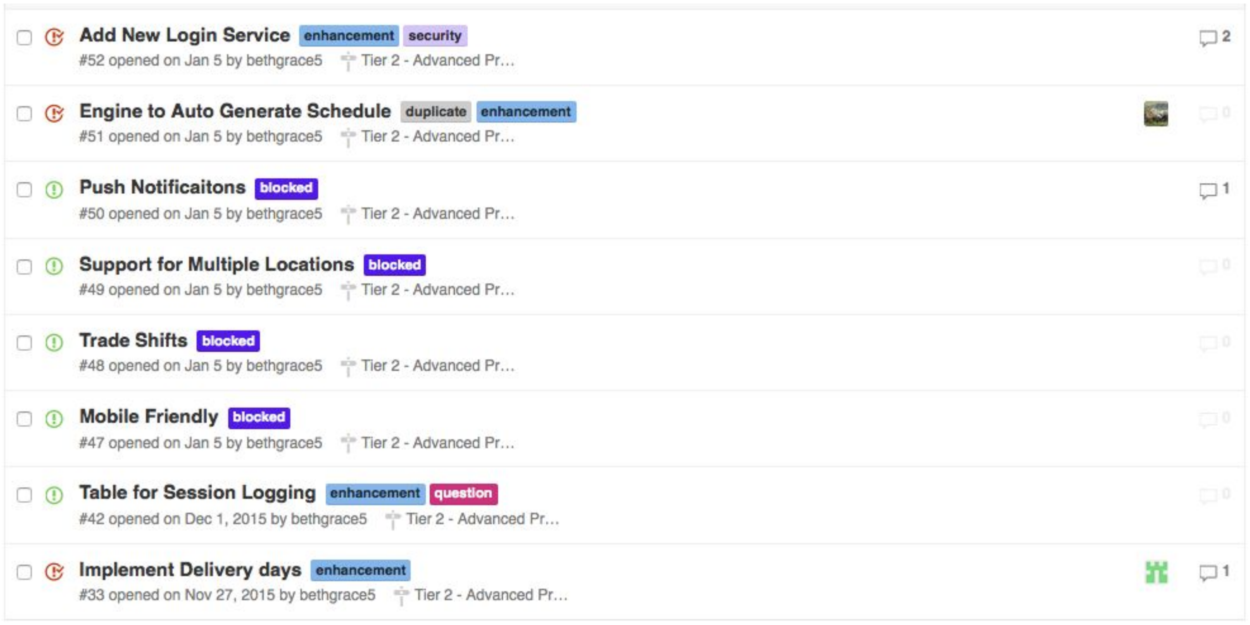
Save Schedules in Browser for efficiency blocked low priority
#30 opened on Nov 27, 2015 by bethgrace5 Tier 1.3 0

redo "Import Last Week" to be import <selected> week, and be more stable of an implementation enhancement
#26 opened on Nov 27, 2015 by bethgrace5 Tier 1.3 0

Schedule Employees WIth Identical First Names - have user select name, or use last initial bug
#21 opened on Nov 9, 2015 by bethgrace5 Tier 1.3 0

Fix The Browser Tab Icon enhancement
#9 opened on Aug 6, 2015 by bethgrace5 Tier 1.3 0

Summary of Tier 2


Add New Login Service enhancement security
#52 opened on Jan 5 by bethgrace5 Tier 2 - Advanced Pr… 2

Engine to Auto Generate Schedule duplicate enhancement
#51 opened on Jan 5 by bethgrace5 Tier 2 - Advanced Pr… 0

Push Notificaitons blocked
#50 opened on Jan 5 by bethgrace5 Tier 2 - Advanced Pr… 1

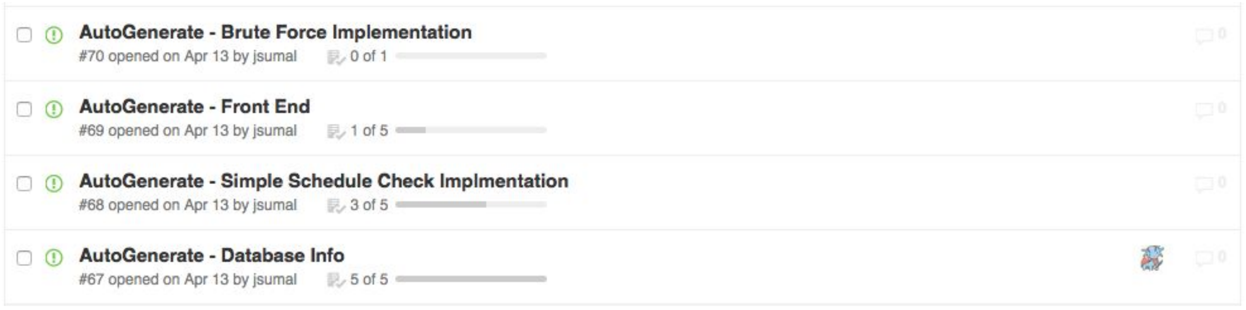
Support for Multiple Locations blocked
#49 opened on Jan 5 by bethgrace5 Tier 2 - Advanced Pr… 0

Trade Shifts blocked
#48 opened on Jan 5 by bethgrace5 Tier 2 - Advanced Pr… 0

Mobile Friendly blocked
#47 opened on Jan 5 by bethgrace5 Tier 2 - Advanced Pr… 0

Table for Session Logging enhancement question
#42 opened on Dec 1, 2015 by bethgrace5 Tier 2 - Advanced Pr… 0

Implement Delivery days enhancement
#33 opened on Nov 27, 2015 by bethgrace5 Tier 2 - Advanced Pr… 1


AutoGenerate - Brute Force Implementation
#70 opened on Apr 13 by jsumal 0 of 1 0

AutoGenerate - Front End
#69 opened on Apr 13 by jsumal 1 of 5 0

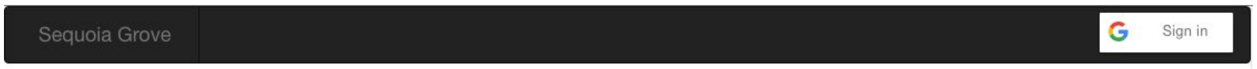
AutoGenerate - Simple Schedule Check Implmentation
#68 opened on Apr 13 by jsumal 3 of 5 0

AutoGenerate - Database Info
#67 opened on Apr 13 by jsumal 5 of 5 0

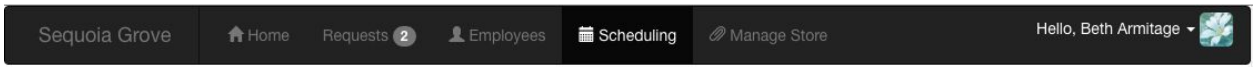
Overview of the final project including screenshots

Header

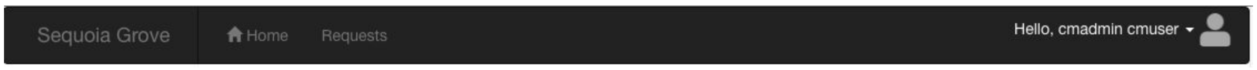
The header allows for navigation of the site. If no user is logged in, it shows no navigation options, and it give a sign in option. If a user is logged in, It displays the current logged in user information retrieved from google and navigation options only available to the user based on their permissions.



Not logged in



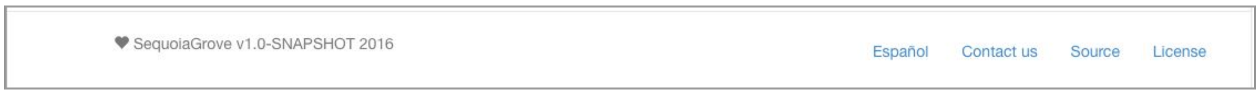
Manager logged in



Employee logged in

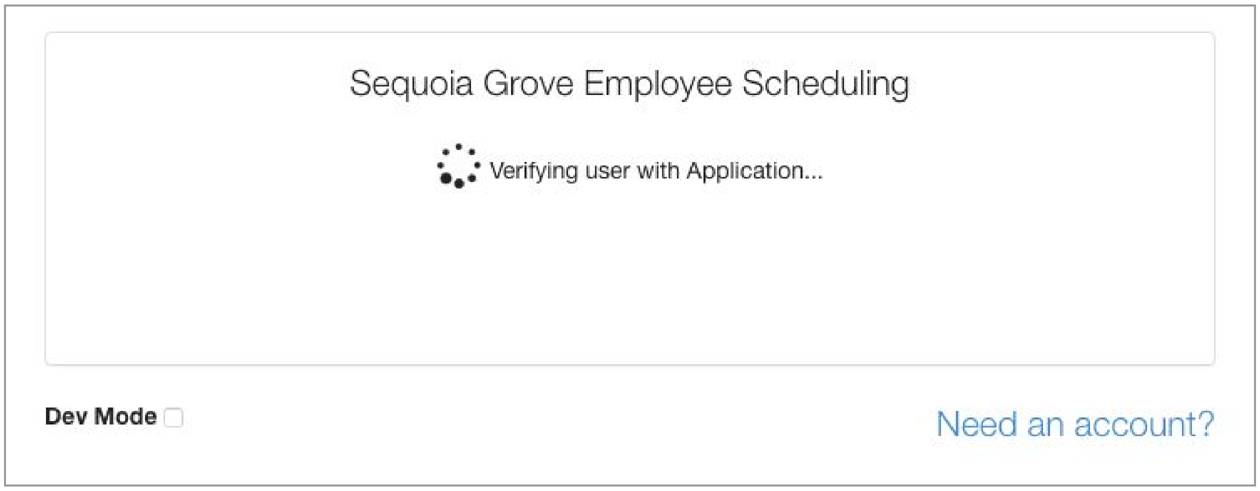
Footer

The footer displays the application version number, an option to translate the page to spanish (which is not a complete site translation yet), a link to google forms to report any problems, a link to the source code, and a link to the product license.



Login View

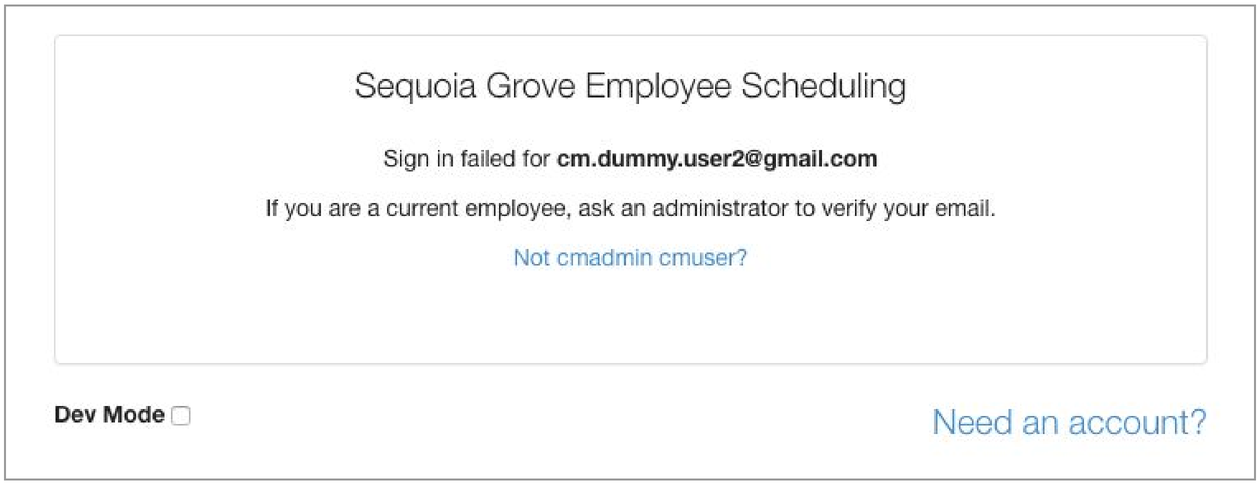
The Login view displays a title for the site, loading progress when logging in, and a message for login failure. It also shows a link for signing up for an account for an organization (under construction).



Sequoia Grove Employee Scheduling

Verifying user with Application...

User logging in



Sequoia Grove Employee Scheduling

Sign in failed for **cm.dummy.user2@gmail.com**

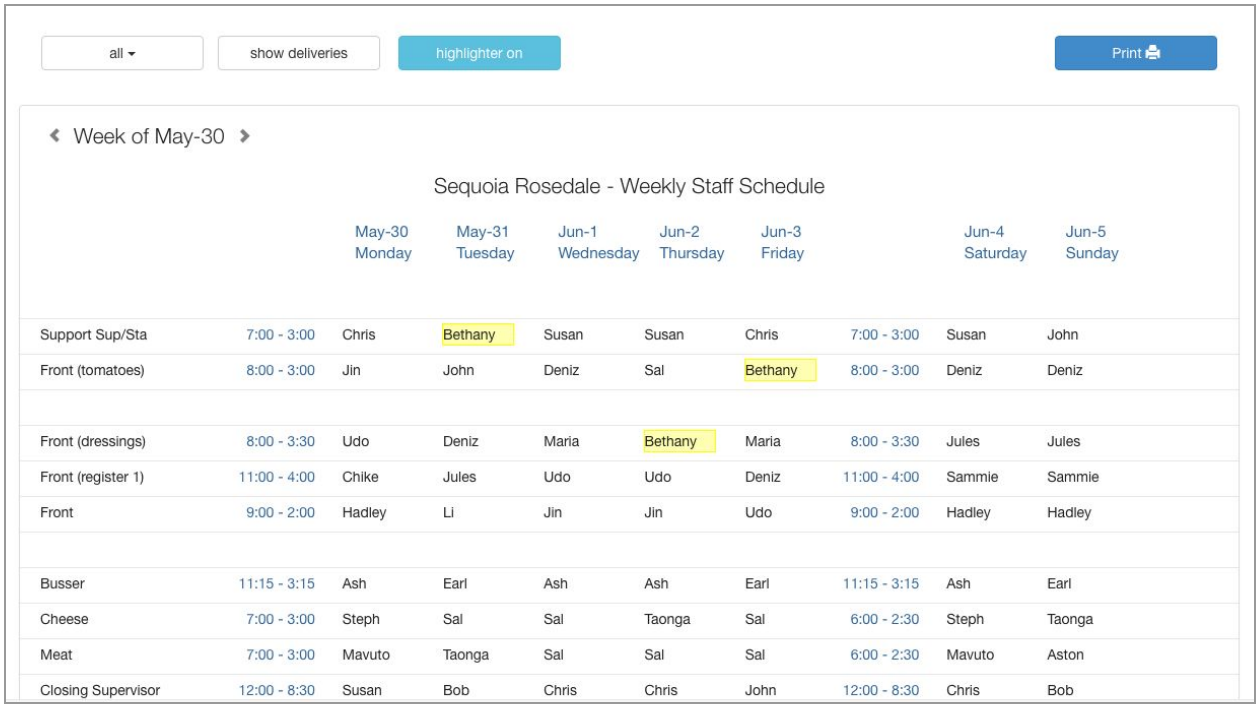
If you are a current employee, ask an administrator to verify your email.

Not cmadmin cmuser?

User login failed

Home View

The home view features an interactive schedule view where the signed in user can navigate weeks by clicking left and right arrow buttons, and highlight their name by the click of a button. The displayed shifts can also be narrowed down by selecting the type of position required, or further narrowed to show ones where the logged in user is scheduled for that shift any day of the week. The user can also toggle the display of recurring deliveries the store will receive, and access a printer friendly page for the schedule.

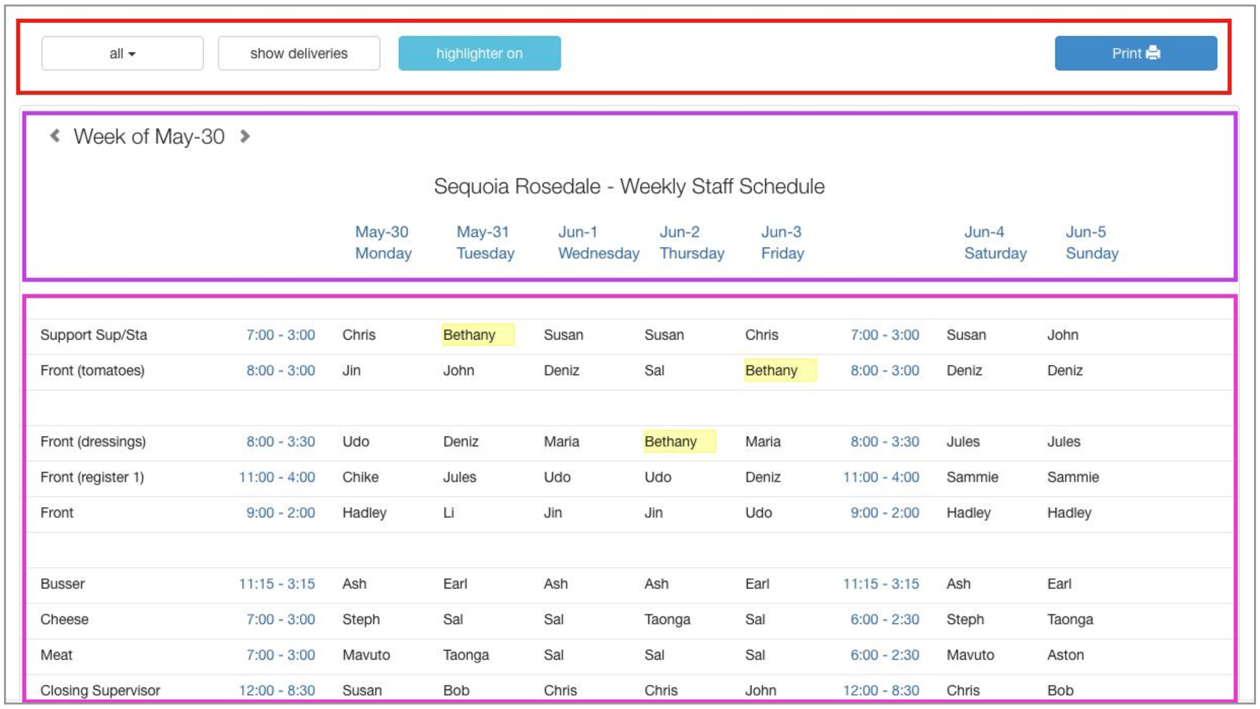


Week of May-30

Sequoia Rosedale - Weekly Staff Schedule

		May-30 Monday	May-31 Tuesday	Jun-1 Wednesday	Jun-2 Thursday	Jun-3 Friday		Jun-4 Saturday	Jun-5 Sunday
Support Sup/Sta	7:00 - 3:00	Chris	Bethany	Susan	Susan	Chris	7:00 - 3:00	Susan	John
Front (tomatoes)	8:00 - 3:00	Jin	John	Deniz	Sal	Bethany	8:00 - 3:00	Deniz	Deniz
Front (dressings)	8:00 - 3:30	Udo	Deniz	Maria	Bethany	Maria	8:00 - 3:30	Jules	Jules
Front (register 1)	11:00 - 4:00	Chike	Jules	Udo	Udo	Deniz	11:00 - 4:00	Sammie	Sammie
Front	9:00 - 2:00	Hadley	Li	Jin	Jin	Udo	9:00 - 2:00	Hadley	Hadley
Busser	11:15 - 3:15	Ash	Earl	Ash	Ash	Earl	11:15 - 3:15	Ash	Earl
Cheese	7:00 - 3:00	Steph	Sal	Sal	Taonga	Sal	6:00 - 2:30	Steph	Taonga
Meat	7:00 - 3:00	Mavuto	Taonga	Sal	Sal	Sal	6:00 - 2:30	Mavuto	Aston
Closing Supervisor	12:00 - 8:30	Susan	Bob	Chris	Chris	John	12:00 - 8:30	Chris	Bob

Home View

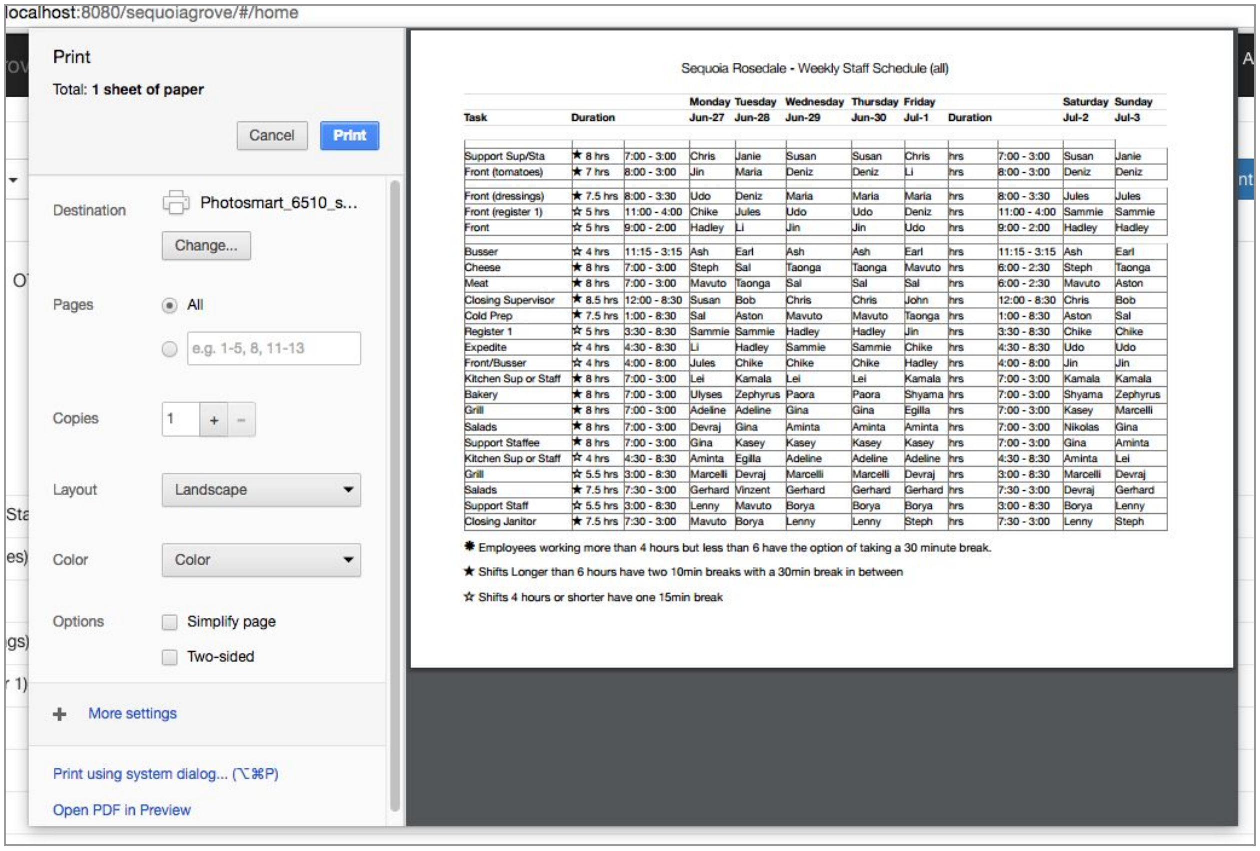


Week of May-30

Sequoia Rosedale - Weekly Staff Schedule

		May-30 Monday	May-31 Tuesday	Jun-1 Wednesday	Jun-2 Thursday	Jun-3 Friday		Jun-4 Saturday	Jun-5 Sunday
Support Sup/Sta	7:00 - 3:00	Chris	Bethany	Susan	Susan	Chris	7:00 - 3:00	Susan	John
Front (tomatoes)	8:00 - 3:00	Jin	John	Deniz	Sal	Bethany	8:00 - 3:00	Deniz	Deniz
Front (dressings)	8:00 - 3:30	Udo	Deniz	Maria	Bethany	Maria	8:00 - 3:30	Jules	Jules
Front (register 1)	11:00 - 4:00	Chike	Jules	Udo	Udo	Deniz	11:00 - 4:00	Sammie	Sammie
Front	9:00 - 2:00	Hadley	Li	Jin	Jin	Udo	9:00 - 2:00	Hadley	Hadley
Busser	11:15 - 3:15	Ash	Earl	Ash	Ash	Earl	11:15 - 3:15	Ash	Earl
Cheese	7:00 - 3:00	Steph	Sal	Sal	Taonga	Sal	6:00 - 2:30	Steph	Taonga
Meat	7:00 - 3:00	Mavuto	Taonga	Sal	Sal	Sal	6:00 - 2:30	Mavuto	Aston
Closing Supervisor	12:00 - 8:30	Susan	Bob	Chris	Chris	John	12:00 - 8:30	Chris	Bob

Home contains schedule-view-options, schedule-view and schedule-header HTML files.


localhost:8080/sequoiagrove/#/home

Print

Total: **1 sheet of paper**

Cancel | Print

Destination: Photosmart_6510_s...

Pages: [✓ All]

Copies: 1

Layout: Landscape

Color: Color

Options: Simplify page, Two-sided

Sequoia Rosedale - Weekly Staff Schedule (all)

			Monday	Tuesday	Wednesday	Thursday	Friday			Saturday	Sunday
Task	Duration		Jun-27	Jun-28	Jun-29	Jun-30	Jul-1	Duration		Jul-2	Jul-3
Support Sup/Sta	★ 8 hrs	7:00 - 3:00	Chris	Janie	Susan	Susan	Chris	hrs	7:00 - 3:00	Susan	Janie
Front (tomatoes)	★ 7 hrs	8:00 - 3:00	Jin	Maria	Deniz	Deniz	Li	hrs	8:00 - 3:00	Deniz	Deniz
Front (dressings)	★ 7.5 hrs	8:00 - 3:30	Udo	Deniz	Maria	Maria	Maria	hrs	8:00 - 3:30	Jules	Jules
Front (register 1)	☆ 5 hrs	11:00 - 4:00	Chike	Jules	Udo	Udo	Deniz	hrs	11:00 - 4:00	Sammie	Sammie
Front	☆ 5 hrs	9:00 - 2:00	Hadley	Li	Jin	Jin	Udo	hrs	9:00 - 2:00	Hadley	Hadley
Busser	☆ 4 hrs	11:15 - 3:15	Ash	Earl	Ash	Ash	Earl	hrs	11:15 - 3:15	Ash	Earl
Cheese	★ 8 hrs	7:00 - 3:00	Steph	Sal	Taonga	Taonga	Mavuto	hrs	6:00 - 2:30	Steph	Taonga
Meat	★ 8 hrs	7:00 - 3:00	Mavuto	Taonga	Sal	Sal	Sal	hrs	6:00 - 2:30	Mavuto	Aston
Closing Supervisor	★ 8.5 hrs	12:00 - 8:30	Susan	Bob	Chris	Chris	John	hrs	12:00 - 8:30	Chris	Bob
Cold Prep	★ 7.5 hrs	1:00 - 8:30	Sal	Aston	Mavuto	Mavuto	Taonga	hrs	1:00 - 8:30	Aston	Sal
Register 1	☆ 5 hrs	3:30 - 8:30	Sammie	Sammie	Hadley	Hadley	Jin	hrs	3:30 - 8:30	Chike	Chike
Expedite	☆ 4 hrs	4:30 - 8:30	Li	Hadley	Sammie	Sammie	Chike	hrs	4:30 - 8:30	Udo	Udo
Front/Busser	☆ 4 hrs	4:00 - 8:00	Jules	Chike	Chike	Chike	Hadley	hrs	4:00 - 8:00	Jin	Jin
Kitchen Sup or Staff	★ 8 hrs	7:00 - 3:00	Lei	Kamala	Lei	Lei	Kamala	hrs	7:00 - 3:00	Kamala	Kamala
Bakery	★ 8 hrs	7:00 - 3:00	Ulyses	Zephyrus	Paora	Paora	Shyama	hrs	7:00 - 3:00	Shyama	Zephyrus
Grill	★ 8 hrs	7:00 - 3:00	Adeline	Adeline	Gina	Gina	Egilla	hrs	7:00 - 3:00	Kasey	Marcelli
Salads	★ 8 hrs	7:00 - 3:00	Devraj	Gina	Aminta	Aminta	Aminta	hrs	7:00 - 3:00	Nikolas	Gina
Support Staffee	★ 8 hrs	7:00 - 3:00	Gina	Kasey	Kasey	Kasey	Kasey	hrs	7:00 - 3:00	Gina	Aminta
Kitchen Sup or Staff	☆ 4 hrs	4:30 - 8:30	Aminta	Egilla	Adeline	Adeline	Adeline	hrs	4:30 - 8:30	Aminta	Lei
Grill	☆ 5.5 hrs	3:00 - 8:30	Marcelli	Devraj	Marcelli	Marcelli	Devraj	hrs	3:00 - 8:30	Marcelli	Devraj
Salads	★ 7.5 hrs	7:30 - 3:00	Gerhard	Vinzent	Gerhard	Gerhard	Gerhard	hrs	7:30 - 3:00	Devraj	Gerhard
Support Staff	☆ 5.5 hrs	3:00 - 8:30	Lenny	Mavuto	Borya	Borya	Borya	hrs	3:00 - 8:30	Borya	Lenny
Closing Janitor	★ 7.5 hrs	7:30 - 3:00	Mavuto	Borya	Lenny	Lenny	Steph	hrs	7:30 - 3:00	Lenny	Steph

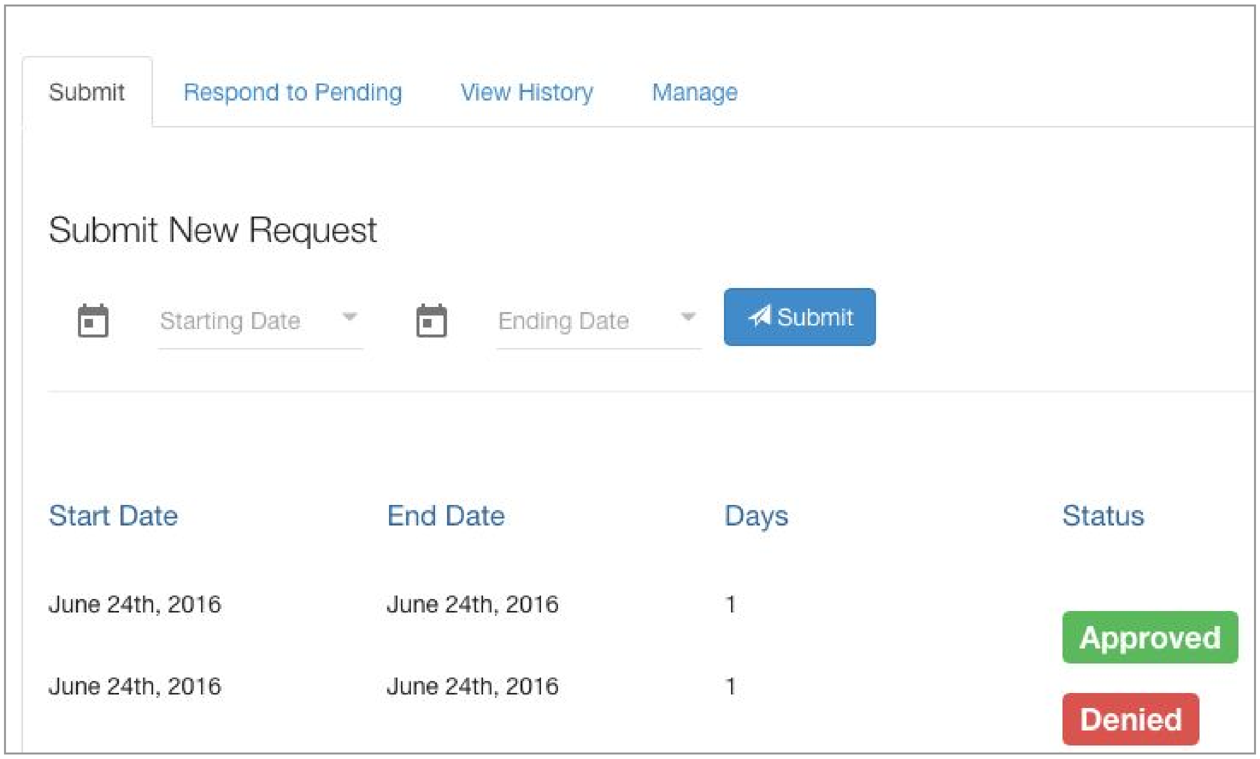
✸ Employees working more than 4 hours but less than 6 have the option of taking a 30 minute break.

★ Shifts Longer than 6 hours have two 10min breaks with a 30min break in between

☆ Shifts 4 hours or shorter have one 15min break

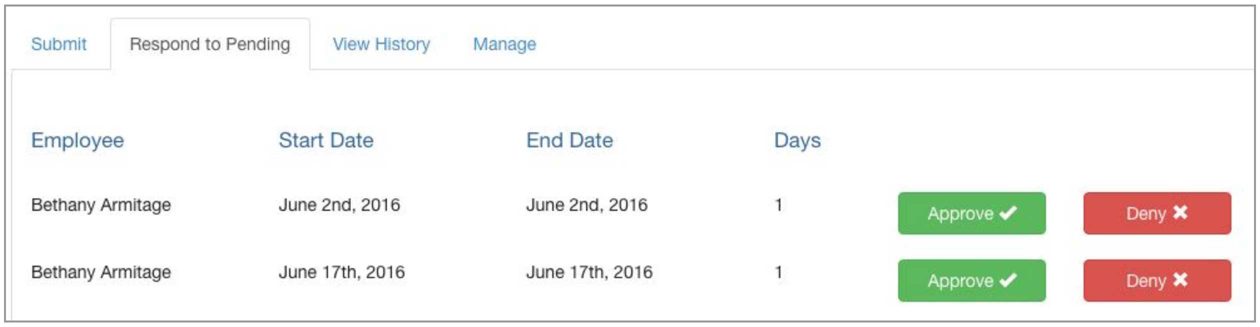
Requests View

A user who has permissions to request vacation may submit vacation requests to reserve days they need free from work. They will select the starting date and end date and click submit to send their request. The user will also see a list of their previous requests and if they are pending, approved or denied. A user with permission manage requests can then approve or deny any pending requests, view the history of all requests, and submit requests for an employee.

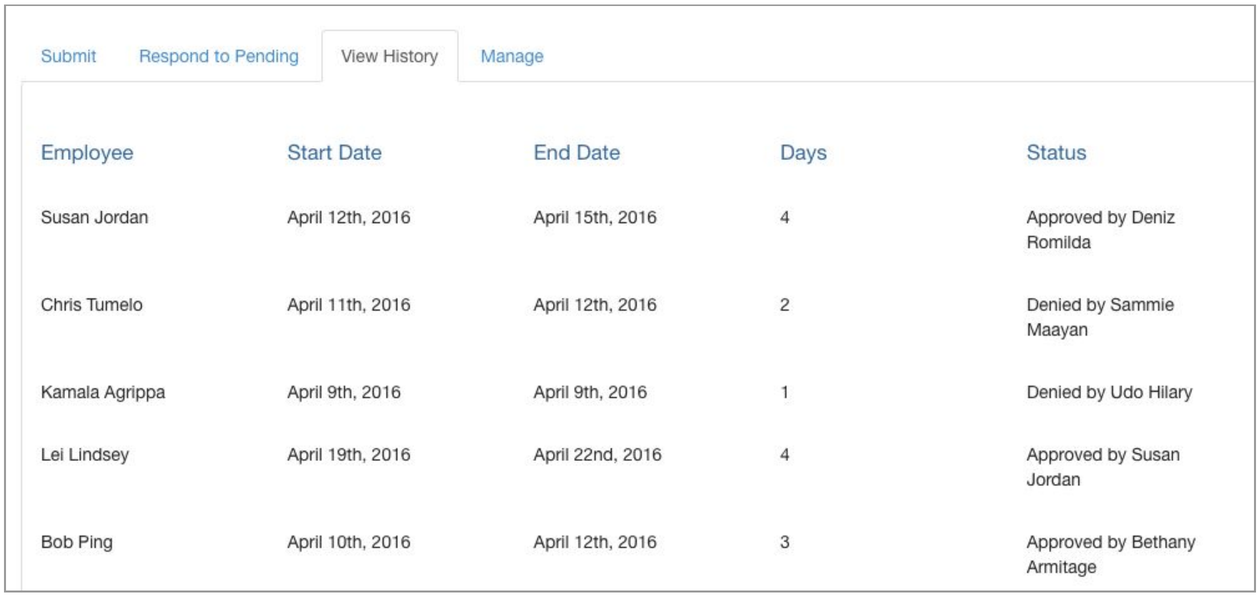


Submit New Request

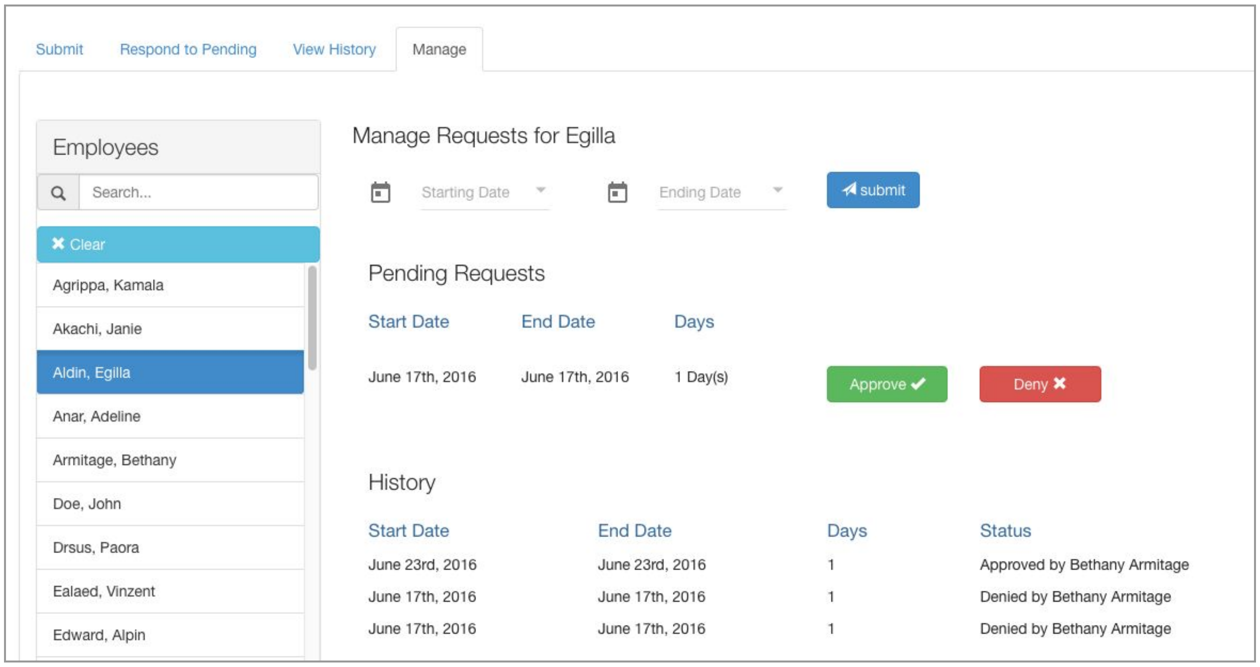
Start Date	End Date	Days	Status
June 24th, 2016	June 24th, 2016	1	Approved
June 24th, 2016	June 24th, 2016	1	Denied



Employee	Start Date	End Date	Days		
Bethany Armitage	June 2nd, 2016	June 2nd, 2016	1	Approve ✓	Deny ✖
Bethany Armitage	June 17th, 2016	June 17th, 2016	1	Approve ✓	Deny ✖



Employee	Start Date	End Date	Days	Status
Susan Jordan	April 12th, 2016	April 15th, 2016	4	Approved by Deniz Romilda
Chris Tumelo	April 11th, 2016	April 12th, 2016	2	Denied by Sammie Maayan
Kamala Agrippa	April 9th, 2016	April 9th, 2016	1	Denied by Udo Hilary
Lei Lindsey	April 19th, 2016	April 22nd, 2016	4	Approved by Susan Jordan
Bob Ping	April 10th, 2016	April 12th, 2016	3	Approved by Bethany Armitage



Manage Requests for Egilla

Pending Requests

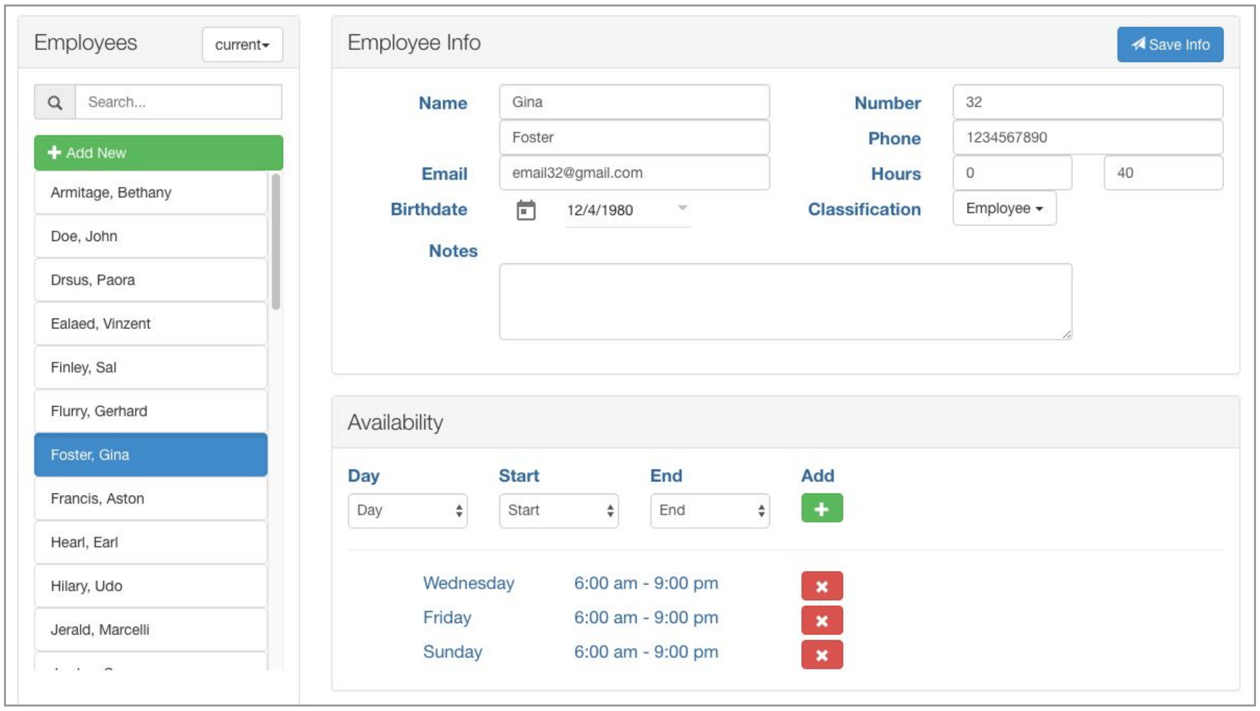
Start Date	End Date	Days
June 17th, 2016	June 17th, 2016	1 Day(s)

History

Start Date	End Date	Days	Status
June 23rd, 2016	June 23rd, 2016	1	Approved by Bethany Armitage
June 17th, 2016	June 17th, 2016	1	Denied by Bethany Armitage
June 17th, 2016	June 17th, 2016	1	Denied by Bethany Armitage

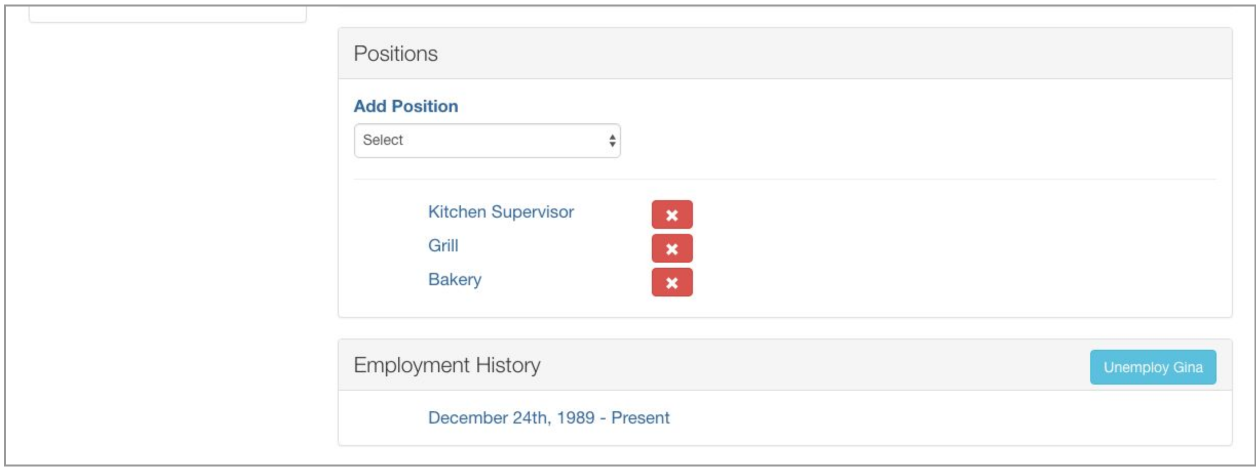
Employee View

Employees may be managed by users with permission manage employees. The manage employee view features a list of employees that can be narrowed to current, past or all employees. Upon selecting an employee from a list, the user can edit their basic information, availability, positions, and view their employment history with the company, and employ or unemploy them.



Name	Gina	Number	32
	Foster	Phone	1234567890
Email	email32@gmail.com	Hours	0 / 40
Birthdate	12/4/1980	Classification	Employee
Notes			

Wednesday	6:00 am - 9:00 pm
Friday	6:00 am - 9:00 pm
Sunday	6:00 am - 9:00 pm



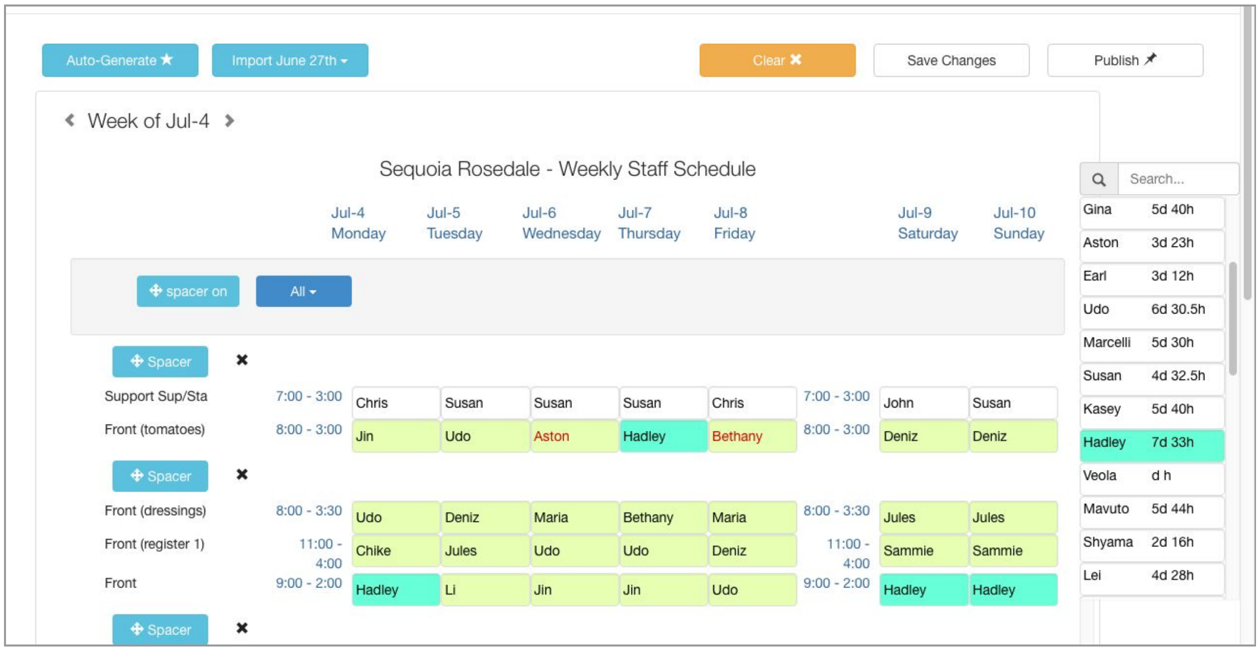
Kitchen Supervisor
Grill
Bakery

December 24th, 1989 - Present

Schedule View

The schedule view features an interactive schedule editing view. It accessible to users with permission manage schedule. Selecting a name on the schedule will highlight all instances of that name so the scheduler can easily scan the distribution of that name. A panel on the right features the names of all current employees, where the name will also be highlighted, and the name can also be selected to highlight all instances of that name. This panel shows the number of days and hours each employee is scheduled, and updates each time the schedule is saved. Selecting an instance of a name on the schedule also highlights all shifts that employee can works, and highlights all employees that can work that shift in the right panel.

The user can perform batch operations such as clearing the entire schedule, and importing a selected week into the current week. A user can publish the schedule when it is ready. The schedule can be narrowed by type of position, and the employee list also will display all employees who are able to work that position. The schedule dynamically makes checks while being edited. If makes sure that the employee being typed in exists within the list of current employees. It also checks availability when a name is added to the schedule.



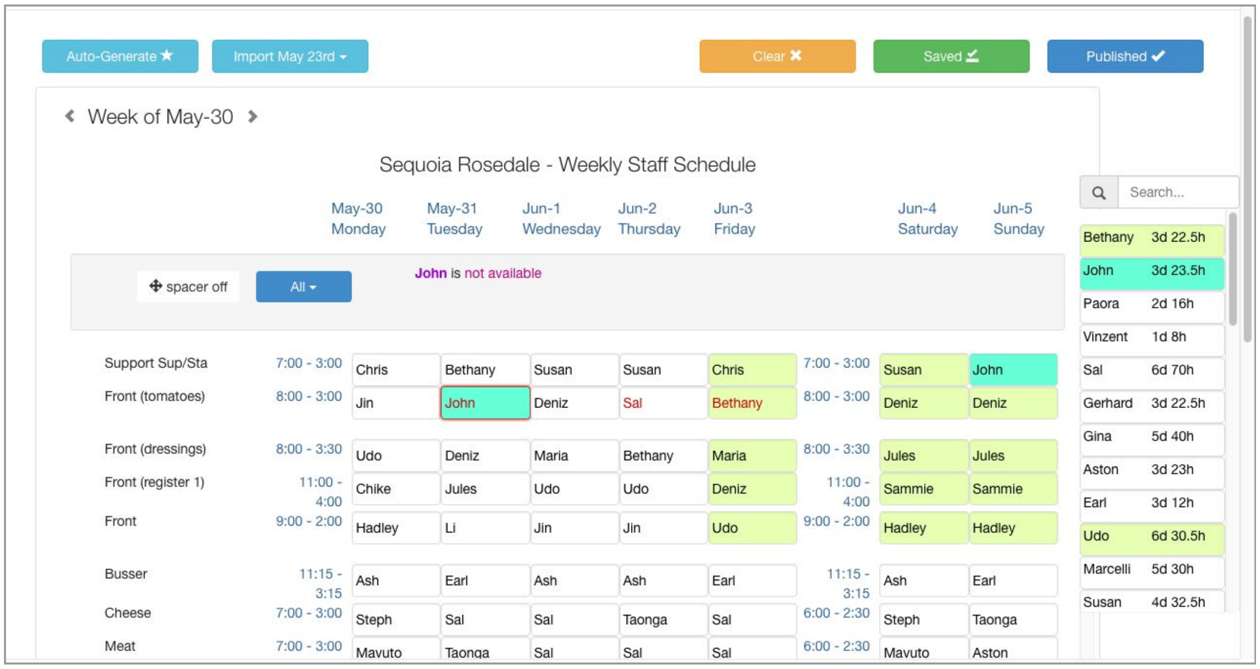
Sequoia Rosedale - Weekly Staff Schedule

		Jul-4 Monday	Jul-5 Tuesday	Jul-6 Wednesday	Jul-7 Thursday	Jul-8 Friday		Jul-9 Saturday	Jul-10 Sunday
Support Sup/Sta	7:00 - 3:00	Chris	Susan	Susan	Susan	Chris	7:00 - 3:00	John	Susan
Front (tomatoes)	8:00 - 3:00	Jin	Udo	Aston	Hadley	Bethany	8:00 - 3:00	Deniz	Deniz
Front (dressings)	8:00 - 3:30	Udo	Deniz	Maria	Bethany	Maria	8:00 - 3:30	Jules	Jules
Front (register 1)	11:00 - 4:00	Chike	Jules	Udo	Udo	Deniz	11:00 - 4:00	Sammie	Sammie
Front	9:00 - 2:00	Hadley	Li	Jin	Jin	Udo	9:00 - 2:00	Hadley	Hadley

Gina	5d 40h
Aston	3d 23h
Earl	3d 12h
Udo	6d 30.5h
Marcelli	5d 30h
Susan	4d 32.5h
Kasey	5d 40h
Hadley	7d 33h
Veola	d h
Mavuto	5d 44h
Shyama	2d 16h
Lei	4d 28h

The schedule edit displays all instances of a selected employee and the positions they can work. It indicates that the schedule has not been saved by showing “save changes” on the save button, and having it not colored. It also indicates that the schedule has not been published by the action “Publish” on the button and it also not having color.
Here, the schedule has all changes saved and is published. It displays relevant errors when a spot is clicked with an error. For the spot clicked in the image, it shows that the selected employee would be available to work for any green space on the schedule. On the other hand,

any employee that is listed in the side list that is green would be available to work for the selected space.



Sequoia Rosedale - Weekly Staff Schedule

John is not available

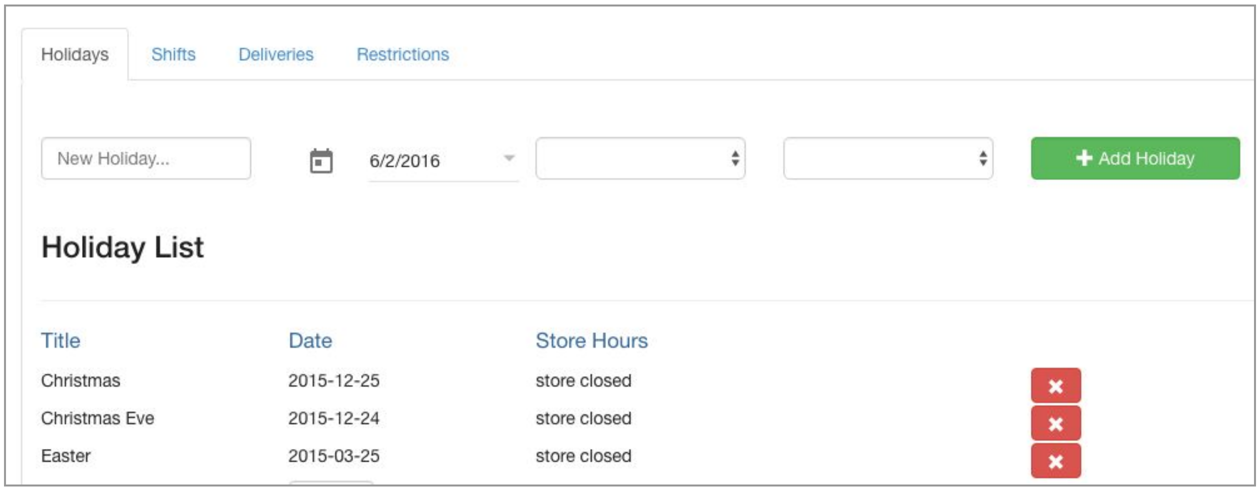
		May-30 Monday	May-31 Tuesday	Jun-1 Wednesday	Jun-2 Thursday	Jun-3 Friday		Jun-4 Saturday	Jun-5 Sunday
Support Sup/Sta	7:00 - 3:00	Chris	Bethany	Susan	Susan	Chris	7:00 - 3:00	Susan	John
Front (tomatoes)	8:00 - 3:00	Jin	John	Deniz	Sal	Bethany	8:00 - 3:00	Deniz	Deniz
Front (dressings)	8:00 - 3:30	Udo	Deniz	Maria	Bethany	Maria	8:00 - 3:30	Jules	Jules
Front (register 1)	11:00 - 4:00	Chike	Jules	Udo	Udo	Deniz	11:00 - 4:00	Sammie	Sammie
Front	9:00 - 2:00	Hadley	Li	Jin	Jin	Udo	9:00 - 2:00	Hadley	Hadley
Busser	11:15 - 3:15	Ash	Earl	Ash	Ash	Earl	11:15 - 3:15	Ash	Earl
Cheese	7:00 - 3:00	Steph	Sal	Sal	Taonga	Sal	6:00 - 2:30	Steph	Taonga
Meat	7:00 - 3:00	Mavuto	Taonga	Sal	Sal	Sal	6:00 - 2:30	Mavuto	Aston

Bethany	3d 22.5h
John	3d 23.5h
Paora	2d 16h
Vinzent	1d 8h
Sal	6d 70h
Gerhard	3d 22.5h
Gina	5d 40h
Aston	3d 23h
Earl	3d 12h
Udo	6d 30.5h
Marcelli	5d 30h
Susan	4d 32.5h

Manage Store View

The Manage Store View is a view accessible to users that have managers permission. It is important to remember that security is implemented with API tokens. Each token contains permissions that allow or restrict users' access to certain information. Users that have 'manage store' permission get access to multiple tabs aimed at facilitating store operation. The Holidays, Shifts, Delivery and Restrictions tabs make up the Manage Store View.

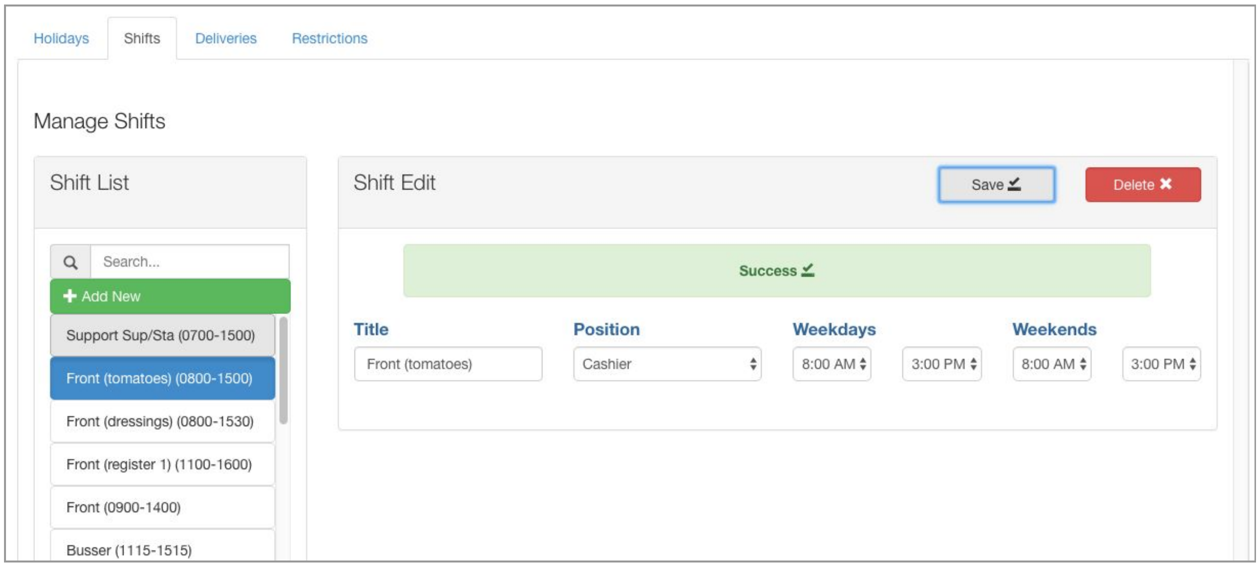
The **Holidays** tab displays a list of holidays during which the store remains closed. Edits can be made to the holidays in two different ways. First, the holidays can be deleted with at the press of a button. A new holiday can also be inserted with drop down boxes to implement a holiday start and end date.



Holiday List

Title	Date	Store Hours
Christmas	2015-12-25	store closed
Christmas Eve	2015-12-24	store closed
Easter	2015-03-25	store closed

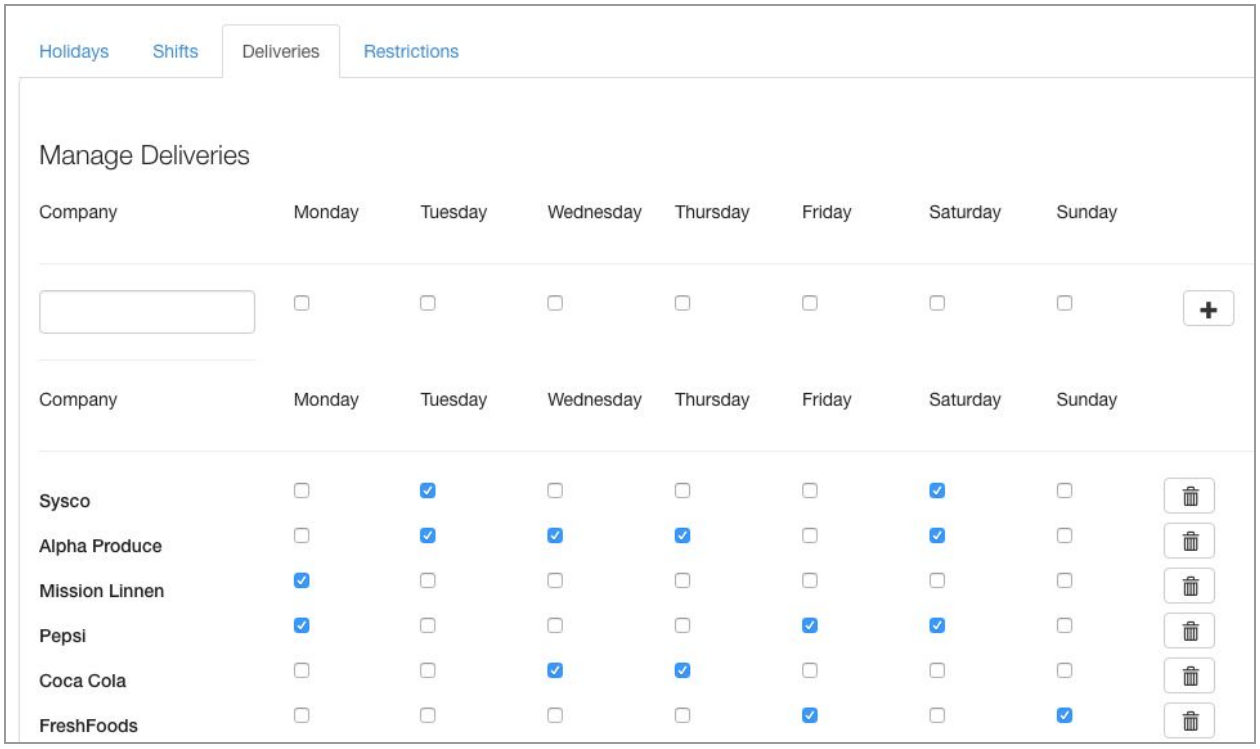
On the **Shifts** tab, supervisors are able to search for shifts on the schedule, new shifts can be added and current shifts can be removed. Shifts titles, positions and weekdays and weekends can also be added.



Support Sup/Sta (0700-1500)
Front (tomatoes) (0800-1500)
Front (dressings) (0800-1530)
Front (register 1) (1100-1600)
Front (0900-1400)
Busser (1115-1515)

Title	Position	Weekdays		Weekends	
Front (tomatoes)	Cashier	8:00 AM	3:00 PM	8:00 AM	3:00 PM

The **Deliveries** tab adds ease to store organization and store logistics. The delivery days for different suppliers are listed for staff to view and prepare accordingly. New suppliers and delivery days can be added and deleted as needed.



Manage Deliveries

Company	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	
	☐	☐	☐	☐	☐	☐	☐	+

Company	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	
Sysco	☐	✓	☐	☐	☐	✓	☐	🗑
Alpha Produce	☐	✓	✓	✓	☐	✓	☐	🗑
Mission Linnen	✓	☐	☐	☐	☐	☐	☐	🗑
Pepsi	✓	☐	☐	☐	✓	✓	☐	🗑
Coca Cola	☐	☐	✓	✓	☐	☐	☐	🗑
FreshFoods	☐	☐	☐	☐	✓	☐	✓	🗑

The **Restrictions** tab is the fourth and final tab under Manage Store View. At this final stage of the project, this tab is under construction and will be completed beyond the scope of the senior project.

Security

Previously, our application used Mozilla Persona as a way to sign in on the client side. Mozilla Persona sent an http request to verify the login credentials of the user. Successful verification validated the user's credentials and allowed access to the application.

With most web companies using an API, tokens are the best way to handle authentication for multiple users. API tokens are scalable and stateless. In other words, a token, not a cookie is sent on every request made from a user. On a security standpoint, a user's access can be controlled, restricted or revoked since token expiration and token revocation can be implemented.

API tokens also allow for an application to be versatile and adaptable. With these security tokens, selective permissions can be granted to third party applications. The API token makes its way from the front to the back end with the help of http interceptors. These are handlers for the request and are involved in the token roundtrip process. From firing off the request process by pulling the token from the local storage to verifying the token in the back end by checking the database and then changing the token on its way back to the front end. The new updated token is then saved to the local storage ready for the next client side request.

Auto Generate

This is major feature of the project designed to enhance the user experience on the ‘Scheduling’ page by helping them construct a schedule for the current week. Unlike importing from the previous week which only copies the previous week into the current, auto-generating the schedule is designed to take the current employee availabilities, approved requests for time off, and several weeks of history into account. This means each shift slot is automatically filled with the employee that worked that shift the most in the prior weeks, is available, and knows the position. This greatly decreases the amount of work needed for preparing the current week’s schedule, as the shift slots are much more likely to be filled with employees able to work the given shifts.

Although we could not finish this feature in the time we had for senior seminar, we did get a good start on it’s implementation. Nearly all of the work for this feature is done in the back end, as the feature is not an enhancement to the user interface but instead a method of sending smarter data to the user. A java Generator class was created in the back end to pull the necessary data from the database and manipulate the data to create a schedule. Other files were modified or added as well to assist or implement the class. At this point the generator does not fill in the schedule on the back end, but it is able to pull the necessary information from the database with a prioritized list of employees for each shift slot. As work continues, the generator will trim those lists based on availability, and then fill in the schedule with the best fit employees, which are the employees that worked each shift the most based on the previous weeks in history.

The UI will also be modified as work continues. Currently the front end only has an 'Auto-Generate' button available that sends hardcoded parameters to the back end for creating the schedule. However menu will also become available, either as a popup after the button click or as a dropdown. This menu will allow the user to modify the number of weeks in history the generator will use for selecting the best fit employee, among other options.

Detailed Code Description

Technical Design

Git github maven2 Amazon Web Services

Git is version control system, and **github** is a remote repository to host projects using git. **Apache Maven** was used as a build tool to compile the Java, run a server for the backend, and apply database changes. **Amazon web services** is used to host the web application and database.

AngularJS bower Bootstrap Java Spring Posgresql

The front end was built using the **AngularJS** frontend web application framework. **Bower** is a web dependency management system used to install javascript libraries. **Bootstrap** is a CSS framework used for styling. **Java Spring** is a backend web application framework. **Postgresql** is the sql language the database was written in, and **PgAdminIII** is the DBMS used to interact with the database.

Liquibase

Liquibase is a database version control system to apply database changes across multiple machines.

Directory structure

The project can be classified into four main categories: **documentation**, **backend**, **database**, and **frontend**. Figure 1 shows the directory structure without individual files inside directories.

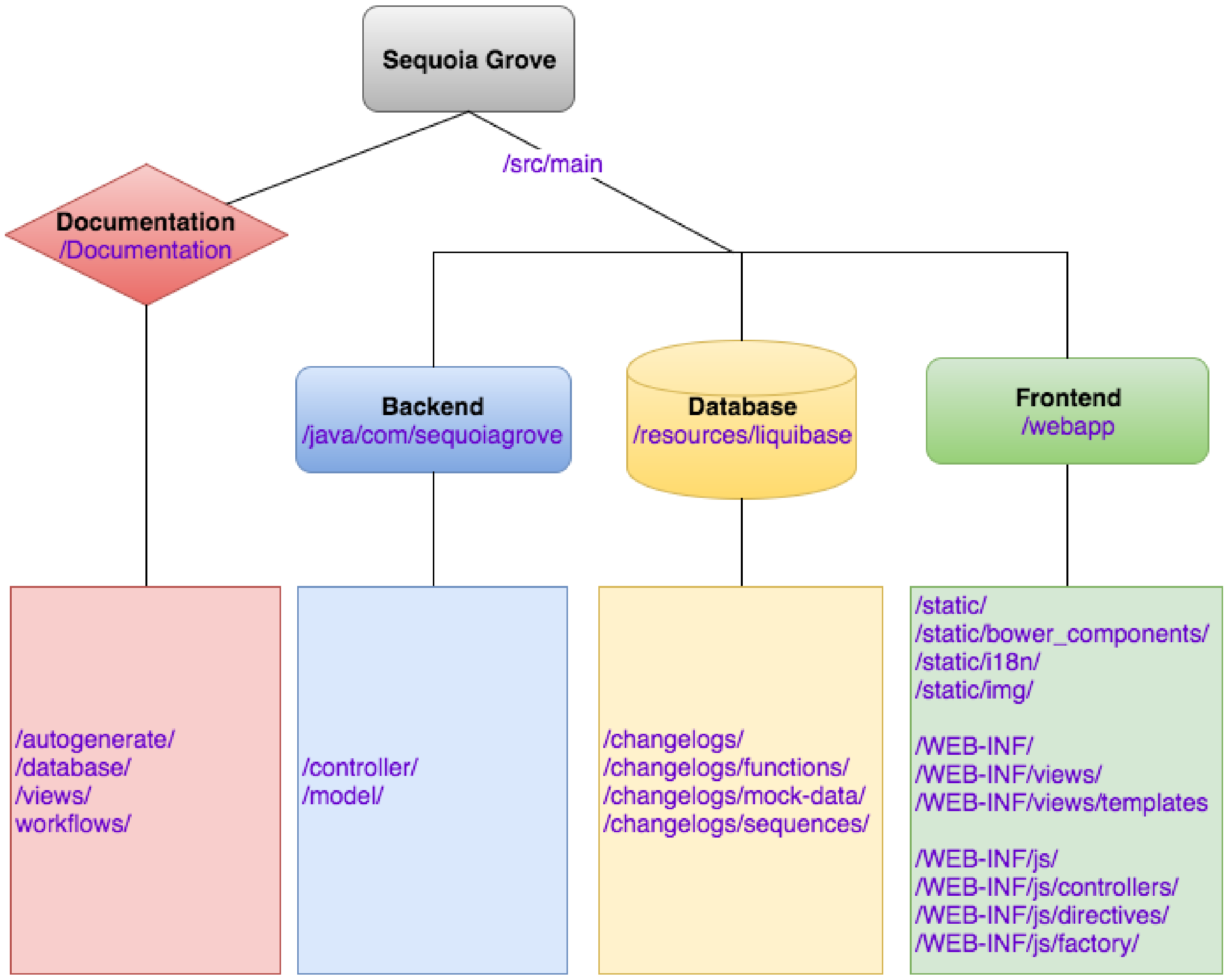


Figure 1

Directory Tree

The following is a comprehensive list of each file in each directory.

```
sequoia-grove
├── README.md
├── bower.json
├── documentation
│   ├── autogenerate
│   │   └── task_list.txt
│   ├── database
│   │   ├── ER_Diagram.pdf
│   │   └── Relational_Database_Diagram.pdf
│   ├── feature-description.md
│   ├── views
│   │   ├── dashboard-employee.png
│   │   ├── dashboard-manager.png
│   │   ├── employee-availibility.png
│   │   ├── employee-edit.png
│   │   ├── login.png
│   │   ├── requests-employee-submit.png
│   │   ├── requests-manager-history.png
│   │   ├── requests-manager-pending.png
│   │   ├── requests-manager-submit.png
│   │   ├── schedule-deliveries.png
│   │   ├── schedule-edit-bottom.png
│   │   ├── schedule-edit-top.png
│   │   ├── schedule-shifts.png
│   │   ├── schedule-view.png
│   │   ├── screenshot-home.png
│   │   ├── screenshot-request-history.png
│   │   ├── screenshot-request-pending.png
│   │   ├── screenshot-request-submit.png
│   │   └── screenshot-schedule-delivery-edit.png
│   └── workflows
│       ├── 000-login.md
│       ├── 001-home.md
│       ├── 002-schedule.md
│       ├── 003-employee.md
│       └── 004-requests.md
│
├── pom.xml
└── src
    └── main
        ├── java
```

```
│        └── com
│            └── sequoiagrove
│                ├── controller
│                │   ├── Authentication.java
│                │   ├── AuthenticationInterceptor.java
│                │   ├── AvailabilityController.java
│                │   ├── DeliveryController.java
│                │   ├── EmployeeController.java
│                │   ├── HolidaysController.java
│                │   ├── MainController.java
│                │   ├── ManageStore.java
│                │   ├── PositionController.java
│                │   ├── RequestController.java
│                │   ├── ScheduleController.java
│                │   └── ScheduleGeneratorController.java
│                └── model
│                    ├── Availability.java
│                    ├── DateCustom.java
│                    ├── Day.java
│                    ├── DayShiftEmployee.java
│                    ├── Delivery.java
│                    ├── Duration.java
│                    ├── Generator.java
│                    ├── Holiday.java
│                    ├── Position.java
│                    ├── PublishSchedule.java
│                    ├── Request.java
│                    ├── RequestStatus.java
│                    ├── ScheduleTemplate.java
│                    ├── Scheduled.java
│                    ├── Shift.java
│                    ├── ShiftRowMapper.java
│                    ├── SuperUserRowMapper.java
│                    ├── User.java
│                    ├── UserRowMapper.java
│                    └── WeeklyAvail.java
├── resources
│   └── liquibase
│       ├── Changelogs
│       │   ├── 001-hours.xml
│       │   ├── 002-employee.xml
│       │   ├── 003-holiday.xml
│       │   ├── 004-employment-history.xml
│       │   ├── 005-availability.xml
│       │   ├── 006-cannot-work-with.xml
│             │       ├── 007-position.xml
│             │       ├── 008-shift.xml
│             │       ├── 009-has-position.xml
│             │       ├── 010-is-scheduled-for.xml
│             │       ├── 011-requests-vacation.xml
│             │       ├── 012-published-schedule.xml
│             │       ├── 013-delivery.xml
│             │       ├── 014-schedule-template-view.xml
│             │       ├── 015-employee-history-view.xml
│             │       ├── 016-schedule-history-view.xml
│             │       ├── 017-schedule-view.xml
│             │       ├── 018-employee-avail-view.xml
│             │       ├── 019-employee-position-view.xml
│             │       ├── 020-employee-info-view.xml
│             │       ├── 021-request-view.xml
│             │       ├── 022-mock-data.xml
│             │       ├── 023-functions.xml
│             │       ├── 024-session.xml
│             │       ├── 025-employee-shift-with-days-view.xml
│             │       ├── 026-get-current-shifts.xml
│             │       ├── 026-rename-user-add-classification-and-permissions.xml
│             │       ├── 027-prefix-tables.xml
│             │       ├──
028-prefix-views-functions-and-sequences.xml
│             │       ├── 029-update-views.xml
│             │       ├── 030-update-functions.xml
│             │       ├── 031-permission-mock-data.xml
│             │       ├── 032-default-test-password.xml
│             │       ├── 033-user-info-notes-birthday-format.xml
│             │       ├── 034-shift-index-column.xml
│             │       ├── 035-user-info-classification.xml
│             │       ├── functions
│             │       │   ├── addShift.sql
│             │       │   ├── delete-schedule.sql
│             │       │   ├── get-current-shifts.sql
│             │       │   ├── get-schedule.sql
│             │       │   ├── publish.sql
│             │       │   ├── schedule.sql
│             │       │   └── update.sql
│             │       ├── mock-data
│             │       │   ├── availability.sql
│             │       │   ├── cannot-work-with.sql
│             │       │   ├── classification.sql
│             │       │   ├── delivery.sql
│             │       │   ├── employee.sql
│       │   │   ├── employment-history.sql
│       │   │   ├── has-position.sql
│       │   │   ├── holiday.sql
│       │   │   ├── hours.sql
│       │   │   ├── is-scheduled-for.sql
│       │   │   ├── permission.sql
│       │   │   ├── position.sql
│       │   │   ├── published-schedule.sql
│       │   │   ├── requests-vacation.sql
│       │   │   └── shift.sql
│       │   └── sequences
│       │       └── sequences.sql
│       ├── db.changelog.xml
│       └── liquibase.properties
└── webapp
    ├── WEB-INF
    │   ├── jdbc.properties
    │   ├── js
    │   │   ├── app.js
    │   │   ├── controllers
    │   │   │   ├── employee.js
    │   │   │   ├── home.js
    │   │   │   ├── login.js
    │   │   │   ├── main.js
    │   │   │   ├── manage.js
    │   │   │   ├── request.js
    │   │   │   └── schedule.js
    │   │   ├── directives
    │   │   │   └── select-on-click.js
    │   │   └── factory
    │   │       ├── auth-token-interceptor.js
    │   │       ├── login-factory.js
    │   │       ├── request-factory.js
    │   │       ├── schedule-factory.js
    │   │       └── user-factory.js
    │   ├── mvc-dispatcher-servlet.xml
    │   ├── styles
    │   │   └── main.scss
    │   ├── views
    │   │   ├── employee.html
    │   │   ├── home.html
    │   │   ├── login.html
    │   │   ├── manage.html
    │   │   ├── request.html
    │   │   ├── schedule.html
    │   │   │
│         │         └── templates
│         │              ├── employee-availibility.html
│         │              ├── employee-history.html
│         │              ├── employee-info.html
│         │              ├── employee-list.html
│         │              ├── employee-permissions.html
│         │              ├── employee-positions.html
│         │              ├── footer-menu.html
│         │              ├── header-menu.html
│         │              ├── manage-deliveries.html
│         │              ├── manage-holidays.html
│         │              ├── manage-restrictions.html
│         │              ├── manage-shifts.html
│         │              ├── request-history.html
│         │              ├── request-manager-mode.html
│         │              ├── request-pending.html
│         │              ├── request-submit.html
│         │              ├── schedule-delivery-edit.html
│         │              ├── schedule-edit-options.html
│         │              ├── schedule-edit.html
│         │              ├── schedule-header.html
│         │              ├── schedule-view-options.html
│         │              ├── schedule-view-print.html
│         │              ├── schedule-view.html
│         │              ├── shift-info.html
│         │              └── shift-list.html
│         └── web.xml
└── static
     ├── i18n
     │    ├── locale-en.json
     │    └── locale-es.json
     ├── img
     │    ├── favicon.ico
     └── index.html
```

Detailed File Descriptions

sequoia-grove/

Project folder, contains documentation, frontend, backend, and database folders

- **README.md:** brief summary of project, purpose, license, external contribution and collaborators.
- **bower.json:** bower, a web package manager, reads this file when the command bower install is run. Each javascript library listed is then installed in each individual development environment.
- **pom.xml:** configuration file for Maven2, a build tool. This file contains configuration for creating a web archive file (.war), a list of java dependencies that maven installs, and project build plugins. This project contains two plugins: the first is for a server called Jetty to run the back end api, the second is for liquibase integration, where it uses the database version control to update individual development environments with any database changes.

/Documentation/

Project documentation and design

- **feature-description.md:** contains descriptions for each feature the project was designed to incorporate, and an incomplete list of screenshots from early on in development.

/Documentation/autogenerate/

Schedule auto generation design

- **task_list.txt:** Schedule auto generation descriptions. Contains task assumptions, plan for technical implementation, possible issues, and user interface description.

/Documentation/database/

Diagrams depicting database schema

- **ER_Diagram.pdf:** Database entity relationship diagram
- **Relational_Database_Diagram.pdf:** Database Relational Diagram

/Documentation/views/

Wireframe design for building user interface

- **dashboard-employee.png**
- **dashboard-manager.png**
- **employee-availibility.png**
- **employee-edit.png**
- **login.png**
- **requests-employee-submit.png**
- **requests-manager-history.png**
- **requests-manager-pending.png**
- **requests-manager-submit.png**
- **schedule-deliveries.png**
- **schedule-edit-bottom.png**

- **schedule-edit-top.png**
- **schedule-shifts.png**
- **schedule-view.png**
- **screenshot-home.png**
- **screenshot-request-history.png**
- **screenshot-request-pending.png**
- **screenshot-request-submit.png**
- **screenshot-schedule-delivery-edit.png**

/Documentation/workflows/

Wireframe design for user experience

- **000-login.md**
- **001-home.md**
- **002-schedule.md**
- **003-employee.md**
- **004-requests.md**

/src/main/java/com/sequoiagrove/controller/

Spring Java API. Each controller listens for incoming requests, calls the function requested, uses JdbcTemplate to interact with the database, and adds attributes added to the org.springframework.ui.ModelMap, which are then accessible in the response.

- **Authentication.java**
  This is used for authentication, it verifies the user by email, verifies their api token, and generates new tokens. It also creates sessions. This file was mainly coded by Beth
  - `getId()[Beth]`, gets the user ID sent from the request
  - `logout()[Beth]`, logs a user out with his user ID, removes session from database
  - `loginWithToken()[Beth]`, checks user token for login
  - `login()[Beth]`, checks user token for login, makes sure it is valid and in the session table
  - `getToken()[Beth]` , creates initial token upon authorization
  - `verifyToken()[Beth]`, checks json web token is valid
  - `nextSessionId()[Beth]`, gets random integer to insert to session table
- **AuthenticationInterceptor.java**
  This handles authentication between the client and the server. This file was coded by Amador and Beth.
  - `preHandle(HttpServeletRequest, Object )[Amador, Beth]`, This is a Pre Handler for before request, it checks if client is authorized from the database.
  - `postHandle ( )[Amador, Beth]`, This is a Post Handler for after request, it checks if there is any errors and changes the status accordingly.
- **AvailabilityController.java**
  This Controller specialize in adding, removing, and modifying the Availability tables in the database upon client request. This file was coded by Beth.
  - `List<String> getPermissions (HttpServletRequest request )[Beth]`, This Function gets a http request from a client and checks if the client have permission to use this controller functions.
  - `String addAvail(Model model, @ModelAttribute(“scope” ) List<String> permissions,@RequestBody String data)[Beth]`, This adds a new Availability data from the client into the Availability table.
  - `String removeAvail(Model model, @ModelAttribute("scope") List<String> permissions, @PathVariable("eid") int eid, @PathVariable("day") String day, @PathVariable("startt") String startt)[Beth]`, This removes a Availability data in the database upon the client request.

- **DeliveryController.java**
  This Controller specialize on adding, removing, and modifying the delivery tables in the database. This file was coded by Amador.
    - `List<String> getPermissions (HttpServletRequest request )[Amador]`, This Function gets a http request from a client and checks if the client have permission to use this controller functions.
    - `public String getDelivery(Model model, @ModelAttribute("scope") List<String> permissions)[AJ]`, This returns all deliveries data in the deliveries table.
    - `public String updateSchedule(@PathVariable ("id") String id, @ModelAttribute("scope") List<String> permissions, Model model)[Amador],` This remove a delivery data in the deliveries table based on the client request.
    - `public String updateDelivery(@RequestBody String data, @ModelAttribute("scope") List<String> permissions, Model model)[Amador],` This updates the Delivery data based on the client request
    - `public String addDelivery(@RequestBody String data, @ModelAttribute("scope") List<String> permissions, Model model)[Amador]`, This adds a new Delivery data into the table based on the client request.
- **EmployeeController.java**
  This Controller specialize on adding, removing, and modifying the delivery tables in the database. This file was coded by Beth and Jasjot.
    - `List<String> getPermissions (HttpServletRequest request )[Beth]`, This Function gets a http request from a client and checks if the client have permission to use this controller functions.
    - `public String getAllEmployee(Model model,@ModelAttribute("scope") List<String> permissions)[Beth]`, This will return a list of employees to the client.
    - `public static WeeklyAvail parseAvailability(String avail)[Jasjot]`, This will parse the string into a java object
    - `public static List<Duration> parseHistory(String hist)[Beth],` This will parse the history string into java objects
    - `public static List<String> parsePositions(String pos) ( )[Beth]`, This will change position string into a java object.
    - `public static List<String> parsePermissions(String permissions)[Beth]`, This will parse the permissions String into a java object.
    - `public String updateEmployee(Model model, @ModelAttribute("scope") List<String> permissions, @RequestBody String data)[Jasjot],` This will update an employee based on the client request.
    - `public String addEmployee(Model model, @ModelAttribute("scope") List<String> permissions, @RequestBody String data)[Beth],` This will add a new employee into the database based the client request.

    - `public String deactivateEmployee(Model model, @ModelAttribute("scope") List<String> permissions, @RequestBody String data)[Beth],` This will deactivate an employee ,(unemployment), upon client request
    -
    - `public String activateEmployee(Model model, @ModelAttribute("scope") List<String> permissions, @RequestBody String data)[Amador],` This will activate an employee ,(can work), upon client request

- **HolidaysController.java**
  This Controller specialize in adding, removing, and modifying the Holidays tables in the database upon client request. This file was coded by Ted.
    - `List<String> getPermissions (HttpServletRequest request )[Ted]`, This Function gets a http request from a client and checks if the client have permission to use this controller functions.

    - `public String getAllHolidays(Model model)[Ted],` This Function returns all list of holidays to the client upon request.

    - `public String sumbitNewHoliday(@RequestBody String data, @ModelAttribute("scope") List<String> permissions, Model model)[Ted],` This function adds a new holiday to the database based on client request.

    - `public String changeRequestDates(@RequestBody String data, @ModelAttribute("scope") List<String> permissions, Model model)[Ted],` This function modifies existing holidays in the database based on client request

    - `public String deleteRequest(@PathVariable("id") int id, @ModelAttribute("scope") List<String> permissions, Model model)[Ted],` This function removes existing holidays from the database based on client request.

- **MainController.java**
  This is the Main Controller, a helps set up other controllers. This file was coded by Beth.
    - `public String goHome(ModelMap model) ( )[Beth]`, this returns a string of the directory of the index page.

    - `public void setDataSource(DataSource dataSource)[Beth],` this returns the data source from the database

    - `public static JdbcTemplate getJdbcTemplate()[Beth],` this returns the jdbcTemplate;

- **ManageStore.java**
  This Controller specialize in adding, removing, and modifying the Shift and hours in the table. This file was coded by Beth and Jasjot.

    - `public List<String> getPermissions(HttpServletRequest request)[Beth],` this returns a list of permissions that allows the use of editing the store

    - `public boolean validateStrings(String... args)[Jasjot]`, this takes dynamic amount of strings and validates them. If the String is empty or null it will return false. Otherwise return true

    - `public int addHours(String startHr, String endHr)[Jasjot],` this modify hours in the store.

    - `public String addShift(@RequestBody String data, @ModelAttribute( "scope" ) List<String> permissions, Model model) [Jasjot],` this gets client information to add a new shift to the database.

    - `public String updateShift(@RequestBody String data, @ModelAttribute("scope") List<String> permissions, Model model)[Jasjot],` this can modify existing shift hours, names, and the days it's added into.

    - `public String deleteShift(@RequestBody String data, @ModelAttribute("scope") List<String> permissions, Model model) [Jasjot],` this can delete existing shifts in the database.

- **PositionController.java**
  This Controller specialize in adding, removing, and modifying the Positions tables in the database upon client request. This file was coded by Beth and Amador.

    - `public List<String> getPermissions(HttpServletRequest request)[Beth],` this gets a list of permission information

    - `public String getPositions(Model model, @ModelAttribute("scope") List<String> permissions)[Beth],` this returns a list of position and its information

    - `public String addPosition(Model model, @ModelAttribute("scope") List<String> permissions, @RequestBody String data)[Beth],` this add a new Position into the database.

    - `public String removePosition(Model model, @ModelAttribute("scope") List<String> permissions, @RequestBody String data)[Beth, Amador],` this receives information from the client to remove a position with the id that matches with the information

- **RequestController.java**
  This Controller specialize in adding, removing, and modifying the Request tables in the database upon client request. This file was coded by Ted.

    - `public List<String> getPermissions(HttpServletRequest request)[Ted],` get permissions contained in the request

    - `public String sumbitRequest(@RequestBody String data, @ModelAttribute("scope") List<String> permissions, Model model)[Ted, Jasjot],` with client information, it adds a new request with the client information to the database.

    - `public String getRequest( @ModelAttribute("scope") List<String> permissions, Model model)[Ted],` returns a list of requests and its information from the database.

    - `public String getCheckedRequest(Model model, @ModelAttribute("scope") List<String> permissions)[Ted],` this returns a list of requests that are marked by a manager user.

- `public String getPendingRequest(Model model, @ModelAttribute("scope") List<String> permissions)[Ted],` this returns a list of requests that are NOT marked by a manager user.

- `public String getCurrentEmployeeRequestl(Model model, @ModelAttribute("scope") List<String> permissions, @PathVariable("eid") int eid) [Ted],` this returns a list of request that were made by the client user.

- `public String updateRequest(Model model, @ModelAttribute("scope") List<String> permissions, @PathVariable("requestID") int requestID, @PathVariable("approverID") int approverID, @PathVariable("is_approve") int is_approve)[Ted],` this modifies existing request information.

- `public String changeRequestDates(@RequestBody String data, @ModelAttribute("scope") List<String> permissions, Model model)[Ted, Jasjot],` this changes the date of the chosen request

- `public String checkStatus(Integer responder, boolean approval),` this gets 2 parameter, an Integer and an boolean. This will return a String depending on the combination. “Pending, Approved, Denied”.

- `public String getRequestInterval(@PathVariable("start") String start, @PathVariable("end") String end, @ModelAttribute("scope") List<String> permissions, Model model)[Ted],` this returns a list of request between the dates the client has chosen.

- **ScheduleController.java**
  This Controller specialize in adding, removing, and modifying the Schedule tables in the database upon client request. This file was coded by Beth and Jasjot.
  - `public List<String> getPermissions(HttpServletRequest request)[Beth],` extracts the scope and the user’s permissions

  - `public String getScheduleTemplate(Model model, @ModelAttribute("scope") List<String> permissions, @PathVariable("mon") final String mon)[Beth]` , returns the current schedule template

  - `public String saveShifts(Model model,  @RequestBody String data, @ModelAttribute("scope") List<String> permissions) [Beth],`Returns a list of current shifts

  - `public String updateSchedule(@RequestBody String data, @ModelAttribute("scope") List<String> permissions, Model model)[Jasjot, Beth],` updates current shifts in schedule template

  - `public String deleteSchedule(@RequestBody String data, @ModelAttribute("scope") List<String> permissions,  Model model)[Jasjot, Beth],` This Delete A Schedule of the client choice.

  - `public String publishSchedule(@RequestBody String data, @ModelAttribute("scope") List<String> permissions,  Model model)[Amador]`, This publish a Schedule of the client choice.

    - `public String checkifPublished( @PathVariable("date") String mon, @ModelAttribute("scope") List<String> permissions,  Model model)[Amador],` This checks the if the schedule is published and returns the answer.

- **ScheduleGeneratorController.java**
The controller contains functions invoked by http request calls from the front end to build and return the auto-generated schedule. This file was coded by Jasjot and Ted.

    - `public String buildGenerator(@RequestBody String data, Model model)[Jasjot, Ted],` This creates a Generator object that stores the history of schedules between dates. The function then sets up everything up to build the schedule by calling trim functions for the generator, creating the schedule template, and then filling and returning the schedule.

    - `String convertDay(Integer value)[Ted],` This function takes and integer value ranged 1-7 and returns a three-letter string representing a day of the week (“mon”, “tue”, etc.)

    - `void generatorBuildList(Generator generator, String data)[Ted]`, This will make a Generator object and make it retrieve past employee schedule information in a given week.

/src/main/java/com/sequoiagrove/model/

POJO (Plain Old Java Object) classes, which contain class member variables, one or more constructors, and getters and setters to access member variables. The exceptions are the RowMapper classes. These contain functions to change a resultset from a SQL query into a java object.

- **Availability.java**: Availability is a Java Object that shows what 2 points of time in a day. This is used for employees to show what time that he or she can work at. This file was coded by Jasot and
Beth.

- **DateCustom.java**: DateCustom is a Java Object that represents a specific date as year, month, and day integers. This file was coded by Jasjot.
Key Function:
    - `String toString() [Jasjot],` Returns the date as a string in format: “yyyy-mm-dd”

- **Day.java**: Day is a Java Object that represents a day on the schedule. It contains the day of the week, the employee name for the day, and the id of the employee for the day. This file was coded by Beth.

- **DayShiftEmployee.java**: DayShiftEmployee is a Java Object that represents an employee who worked in a shift in a certain day. This is used for the Generator Object. This file was coded by Ted.

- **Delivery.java**: Delivery is a Java Object that represents the delivery information. This file was coded by Amador.

- **Duration.java**: Duration is a Java Object that represents an interval of time, either as start and end dates or start and end times. Constructors are in place to take day or time values as strings are parse them as dates or times. Parsing for times occurs in the constructors, since all times are acquired from the database as integers in 24 hour format, making parsing simple. Parsing dates however required parsing into the DateCustom class made for this application, so our own parse function was made. This file was coded by Beth.
  Key Function:
    - `Public DateCustom parseDateString(String datestr) [Jasjot],` Takes a date string in the format “yyyy-mm-dd” and converts it to a DateCustom object

- **Generator.java**
  The Generator is a java Object which dynamically gathers past schedule information and represent them as a 3 dimensional hashmap. It contains functions that help modify and trim information in the 3D hashmap. This file was coded by Ted and Jasjot.

  3D Hashmap Format:

Key:
`<String> Day`
 	
(Example: “mon”, “tue”, etc)
	Value:
	2D HashMap
	Key:
	`<Int>`
	`Shift ID`
		Value:
		1D HashMap
		Key:
		`<Int>`
		`Employee ID`
			Value:
			`<Int>`
			Number of weeks in history the employee worked this shift on this day

    - `Generator (String mon, final String historyStart, final String historyEnd) [Ted, Jasjot]`, Initialize the generator object and use the parameters to set where to get past schedule information. These parameters are used to make queries to the database and acquire the necessary information needed to generate a schedule.

- `Public void addDay(String day) [Ted]`, Adds the day ("mon", "tue"...) to the 3D hashmap with empty values

- `Public void addShift(String day, Integer shift) [Ted]`, Adds a shift ID to the given day to the 3D hashmap with empty values

- `Public void addEmployee(`
  ```
  String  day,
  Integer shift,
  Integer employee,
  Integer amount)[Jasjot, Ted]
  ```
  , Adds an employee to the given day and shift with the given value in 'amount'

- `Public void add(`
  ```
  String  day,
  Integer shift,
  Integer employee,
  Integer amount)[Ted]
  ```
  , Runs addDay(), addShift(), and addEmployee() with the given values

- `Public void fillGenerator()[Jajot, Ted]`, This will fill the generator using add() with the values in dayShiftEmployeeList.

- `public List<DayShiftEmployee> getPastInformation(String startDate, String endDate)[Jajsot, Ted]`,This function will get past schedule information from the database between the 2 dates in the parameters

- `Public List<User> getEmployeeInformation() [Jasjot]`, This function will query the database for all necessary Employee information, including their availabilities

- `Public List<Shift> getShiftInformation(String mon) [Jasjot]`, This will get shift information from the database on the current week

- `Public List<Request> getRequestInformation(String startDate)[Jasjot, Ted]`, This will query the database for any requests-off approved for the current week

- `Public String checkStatus(Integer responder, boolean approval)[Ted]`, This checks the status of the request, if approved true otherwise false.

- `Public WeeklyAvail parseAvailability(String avail)[Beth]`, This will parse a long list and convert it to a WeeklyAvaility object.

- `Public List<Duration> parseHistory(String hist)[Beth]`, This will change the History string into a list of java objects

- `Public List<String> parsePositions(String pos)[Beth]`, This will parse the Position string into a list of java object

- `Public void trimByListRestriction()[Ted]`,, This will trim the the schedule generator by employee who can't work together.

- `Public void trimByRestriction(String person1, String person2)[Ted]`, This checks if the employees in the parameter are in the same work schedule.

- `Public void trimByUnavaliablity(String person1, String date)[Ted]`, This removes the given employee if his or her work hours don't work for that day or time.

- `Public boolean checkAvaliablity(User user, Shift shift, String dayKey) [Ted]`, This checks if the employee can work the given shift on the given day.

- `Public void trimByRequest()[Ted]`, This removes employees from the 3D hashmap for the days they have requested off and have been approved to take off.

- `Public void removeEmployee(Integer day, Integer shift, Integer employee)[Ted]`, This remove an employee based on the day keys, shift keys, and employee id.

-
  ```
  Public boolean checkEmployeeIf(
         String  day,
  ```

`Integer shift,`
`Integer employee1,`
`Integer employee2)[Ted]`, Checks if employee are in the same work schedule.

- `Int getShiftWithEmployee(String name, String dayKey)[Ted]`, This checks if the shift has the employee
- `Void searchEmployee(String name, String dayKey)[Ted]`, This searches for an employee on based on the dayKey.
- `Public void printFormation()[Ted, Jasjot]`, This prints out the 3d generator information in a neat way. Prints the schedule of workers..

- **Holiday.java**
  Holiday is a java Object that represents the Holidays. It has a date and title and has set hours of what time the store should be open and close on. This file was coded by Ted.

- **Position.java**
  Position is a java Object that represents the position in the work such as Cashier or Janitor. This file was coded by Beth.

- **PublishSchedule.java**
  PublishSchedule is a java Object that represents the publish date for the schedule and the user who published it. This file was coded by Amador.

- **Request.java**
  Request is a java Object that represent request from an user. Request is what user use to submit for days off. It contains the user identity and the dates the user wants for days off. This file was coded by Ted.

- **RequestStatus.java**
  RequestStatus is a java Object that represents A Vacation Request's Status, it is a more detailed object then Request. It is what employees use to submit for days off. It contains the the Request basic information, requester information, approver information, and the status of the request. This file was coded by Ted.

- **ScheduleTemplate.java**
  Schedule Template is a java Object that represents the Schedule information. This file was coded by Beth.

- **Scheduled.java**
  Scheduled is a java object that represents an employee who will work for a shift in a certain day. This file was coded by Beth.

- **Shift.java**
  Shift is a java object that represents the different shifts work. This file was coded by Jasjot.

- **ShiftRowMapper.java**
  This is detailed java object that helps format database information into java objects. This file was coded by Jasjot.
  - `Shift mapRow(ResultSet rs, int rowNum)` [Jasjot], This Is used to help convert information from the database into java objects.

- **SuperUserRowMapper.java**
  This is detailed java object that helps format database information into java objects. This file was coded by Beth and Amador.
  - `User mapRow(ResultSet rs, int rowNum)` [Beth], This Is used to help convert information from the database into java objects.

- **User.java**
  This is a java object that represents users of the application This file was coded by Amador.
  - `Public user(){}` [Amador], default constructor for the User class

  - `Public user(int, int, int, int, String, String, String, String, String, list<Duration>, list<String>, WeeklyAvail, boolean, List<String>, int, String, String){}` [Amador], constructor for weekly availability using all the attributes that make up an availability

  Setters and getters setting and extracting the value of every member are also part of this file.

- **UserRowMapper.java**
  This is a java object that help format database information into java objects. This file was coded by Beth and Amador.

    - `User mapRow(ResultSet rs, int rowNum)` [Beth], This helps convert database entries into java object

- **WeeklyAvail.java**
  This is a java object that represents each day in a week, with each day having its own Duration. This file was coded by Beth and Jasjot.
    - `Public WeeklyAvail(){}` [Jasjot], default constructor for weekly availability
    - `Public WeeklyAvail(mon, tue, wed, thu, fri, sat, sun){}` [Jasjot], constructor setting availability for each weekday

  This file also comprises setters and getters, setting and retrieving the value of the all seven weekdays and their availability.

/src/main/resources/liquibase/

Database version control implementation. The database tracks which changelogs listed in db.changelog.xml have been successfully run. If there are new changelogs in the list it will run them on application startup.

- **db.changelog.xml:** list of changelog files. Upon each application build, liquibase checks which changelogs in the list have been run, and then runs any new additions.
- **Liquibase.properties:** databse url and credentials

/src/main/resources/liquibase/changelogs/

Changelog files for maintaining the database. Changelogs are never updated. They build on top of eachother. For example, if a table needs an additional column, a changelog is added to the list with a statement to add the column to the appropriate table. When a branch is merged to github, database changes are then transported with the corresponding code changes.

- **001-hours.xml:** base table
- **002-employee.xml:** base table
- **003-holiday.xml:** base table
- **004-employment-history.xml:** base table

- **005-availability.xml:** base table
- **006-cannot-work-with.xml:** base table
- **007-position.xml:** base table
- **008-shift.xml:** base table
- **009-has-position.xml:** base table
- **010-is-scheduled-for.xml:** base table
- **011-requests-vacation.xml:** base table
- **012-published-schedule.xml:** base table
- **013-delivery.xml:** base table
- **014-schedule-template-view.xml:** view
- **015-employee-history-view.xml:** view
- **016-schedule-history-view.xml:** view
- **017-schedule-view.xml:** view
- **018-employee-avail-view.xml:** view
- **019-employee-position-view.xml:** view
- **020-employee-info-view.xml:** view
- **021-request-view.xml:** view
- **022-mock-data.xml:** includes path to sql files with mock data
- **023-functions.xml:** includes path to sql files with functions
- **024-session.xml:** base table
- **025-employee-shift-with-days-view.xml:** view
- **026-get-current-shifts.xml:** includes path to sql files with function
- **026-rename-user-add-classification-and-permissions.xml:** various updates to integrate security implementation
- **027-prefix-tables.xml:** renames tables using prefixes for standardization
- **028-prefix-views-functions-and-sequences.xml:** renames views, function and sequences using prefixes for standardization
- **029-update-views.xml:** add parameters to views
- **030-update-functions.xml:** update functions after view changes
- **031-permission-mock-data.xml:** includes path to sql files with mock data
- **032-default-test-password.xml:** sets all passwords to sha-512 encryption of ‘1234’ for mock data
- **033-user-info-notes-birthday-format.xml:** add columns to views
- **034-shift-index-column.xml:** add column to shift table for index
- **035-user-info-classification.xml:** update user info view, renames classification columns for title and id

/src/main/resources/liquibase/changelogs/functions/

SQL files containing database functions

- **addShift.sql:** function to add a shift to the database
- **delete-schedule.sql:** function to remove a name from a scheduled day and shift
- **get-current-shifts.sql:** function to retrieve shifts for the current week
- **get-schedule.sql:** function to build a schedule returned
- **publish.sql:** function to mark a schedule as published
- **schedule.sql:** function to add a name to a day and shift
- **update.sql:** rename functions with prefix

/src/main/resources/liquibase/changelogs/mock-data/

SQL files containing insert statements to build mock data

- **availability.sql**
- **cannot-work-with.sql**
- **classification.sql**
- **delivery.sql**
- **employee.sql**
- **employment-history.sql**
- **has-position.sql**
- **holiday.sql**
- **hours.sql**
- **is-scheduled-for.sql**
- **permission.sql**
- **position.sql**
- **published-schedule.sql**
- **requests-vacation.sql**
- **shift.sql**

/src/main/resources/liquibase/changelogs/sequences/

SQL file containing sequences for primary key generation

- **sequences.sql:** each sequence associated with

/src/main/webapp/WEB-INF/

Directory containing front end files. Contains configuration, HTML, Javascript and CSS files

- **jdbc.properties:** database credentials, not under version control
- **mvc-dispatcher-servlet.xml:** java spring application settings and relative routing and definition of request intercepted routes
- **web.xml:** java spring application settings

/src/main/webapp/WEB-INF/js/

Directory containing javascript angular modules

- **app.js**
  Configuration for angular as the main entry point. Url routing definitions, Initial configuration for dependencies, run block that inserts google signin button so it renders when Angular is ready. This file was coded by Beth.

/src/main/webapp/WEB-INF/js/controllers/

Angular Controllers correspond to views

- **employee.js**
  controller for actions within the employee page. Contains functionality for the employee edit form. This file was coded by Beth and Jasjot.
  - `$scope.changeType()` [Beth], switch filter of employee list type for all, current or past
  - `$scope.filterByType()` [Beth], filter employee list by all, current or past employees
  - `$scope.formatEmploymentHistory()` [Jasjot], formats the date string
  - `$scope.setNewAvailTimes()` [Beth], sets available times start and end
  - `$scope.selectClassification()` [Beth], sets the index of the selected classification

    - `$scope.selectEmployee()` [Beth], verifies that the correct employee was selected
    - `$scope.formatEmploymentHistory()` [Jasjot], formats the date string
    - `$scope.clearEmployee()` [Beth], clears all employee attributes
    - `$scope.addAvailability()` [Beth, Jasjot], adds a new availability time for an employee
    - `$scope.clearEmployee()` [Jasjot], clears all employee attributes
    - `$scope.addPermission()` [Beth], adds a new permission
    - `$scope.removePermission()` [Beth], removes employee permission
    - `$scope.getPositionTitle()` [Beth], gets title for the position
    - `$scope.clearEmployee()` [Beth], clears all employee attributes
    - `$scope.addPosition()` [Beth, Jasjot], adds a new position for an employee
    - `$scope.removeAvailability()` [Jasjot], removes an availability for an employee
    - `$scope.removePosition()` [Beth, Jasjot], removes position from employee front and back end
    - `$scope.updateEmployee()` [Beth, Jasjot], updates existing employee or adds new
    - `$scope.deactivateEmployee()` [Beth], un-employs an employee
    - `$scope.activateEmployee()` [Beth], activates or re-employs an employee

- **home.js**
  Controller for the home view. Contains functionality for the schedule view. Narrows the schedule view by selected type, and controls whether the logged in employee name is set to be highlighted. This file was coded by Beth.
    - `$scope.filterByType (loc, pos, t )` [Beth], This function filters the schedule by the selected position when selected from the menu on the schedule view page.

- **login.js**
  Controller for user login and logout procedures. Handles actions from google signin. This file was coded by Beth.
    - `onSignIn (googleUser )` [Beth], This function is triggered when a user successfully signed in with google, it then passes the authentication to the application, and once successful, kicks off initialization of data throughout the application. If authentication is not successful, it handles the failed login attempt.
    - `signOut ( )` [Beth], This function is triggered when a user signs out with google for this application. It resets the application to a clean state.
    - `switchUser ( )` [Beth], This function logs a user completely out of google (not just google for this application), this is required for switching user accounts because otherwise, the application would simply keep signing in the same user upon page refresh.

    - `initializeData (isManager )` [Beth], This function calls initialization of all parts of the application after the user signs in, it loads information based on whether the user is a manager or not.

- **main.js**

  Main front end controller. All other controllers are contained within this scope. This controller originally was used for sharing data between other controllers. After better practice was learned to share data using the injection of factories, much of the logic was transferred over to factories. However, there is still lingering resolves to the above issue. This file was coded by All members of the team.

    - `getPositions ( )` [Beth], this function retrieves a list of all positions from the database, used for assigning positions to employees as well as listing positions to narrow the view down with.
    - `getDeliveries ( )` [Amador], this function retrieves a list of all expected weekly deliveries. And organizes the result into a format easier iterable by the view showing the deliveries.
    - `initAvailSchedule ( )` [Beth], this function filters schedule to determine if each employee has availability for the given day. It adds 'available' attribute to that day on the schedule template for error checking when scheduling. It also removes availability if request for vacation was approved
    - `initPositionsSchedule ( )` [Beth], this function filters the schedule to determine if each employee has the position for any given day. It adds ‘hasPosition’ attribute to that day on the schedule template for error checking when scheduling.
    - `initIsCurrentSchedule ( )` [Beth], this function adds ‘isCurrent’ attribute to each day on the schedule template showing if each employee is current or not for error checking when scheduling.
    - `publishSchedule ( )` [Amador], this function is a link to the schedule factory to publish a schedule by sending the data to be inserted in the database that will mark this schedule as published.
    - `updateChangesmade ( )` [Beth], this function is called when the Schedule Factory notifies that changes were made. It then fires updates for all information regarding the schedule.

- **manage.js**

  This controller contains functionality for the manage store view. It has functions for editing holidays, shifts, and weekly deliveries. This file was coded by all members of the team.

    - `cleaupShiftEdit ( )` [Jasjot], this function resets the shift form.

  - `verifyShift ( )` [Jasjot], this function verifies that all information was supplied before sending shift.
  - `updateShift ( )` [Jasjot], this function sends a shift to the database when it was changed.
  - `addShift ( )` [Jasjot], this function adds a new shift to the database.
  - `deleteShift ( )` [Jasjot], this function removes a shift from the database.
  - `deleteDelivery ( )` [Amador], this function deletes a delivery from the database.
  - `addDelivery ( )` [Amador], this function adds a new delivery to the database.
  - `compareData ( )` [Ted], this function compares dates
  - `addNewHoliday ( )` [Ted], this function adds a new holiday to the database
  - `getAllHolidays ( )` [Ted], this function retrieves a list of all holidays from the database
  - `deleteHoliday ( )` [Ted], this function deletes a holiday from the database
- **request.js:** gives the all the functionality of adding, removing, and modifying requests for users. This file was coded by Ted.
  - `$scope.updateEnd() ( )` [Ted], this will update the date interface
  - `$scope.isActive(tabName) ( )` [Ted], This will check if the request page is currently active
  - `$scope.totalDays(a, b)` [Ted], This will check the total days between dates
  - `$scope.defaultDate(a)` [Ted], This will return the current date
  - `$scope.getCurrentEmployeeRequest()` [Ted], Get All Employees who are working
  - `$scope.submitRequest()` [Ted], This will submit a request based on the user inputs
  - `$scope.confirmSubmit(ev)` [Ted], This will confirm the submit request
  - `$scope.datesCollidePopup(ev)` [Ted], A popup on the screen will show if dates collide with existing dates.
  - `$scope.checkDatesCollide()` [Ted], This will check if dates collide
  - `$scope.allRequests()` [Ted], This will get all Requests in the database.
  - `$scope.getPendingRequests()` [Ted], This will get all pending requests in the database
  - `$scope.targetEmployee(employee)` [Ted], This is the selected employee for modifying a request for
  - `$scope.targetRequest(request)` [Ted], This will get the identity of the Request.
  - `$scope.managerSubmitRequest()` [Ted], This is the manager to submit request for other users.
  - `$scope.changeRequest($requestID, $approverID, $is_approve)` [Ted], This is the will modify the request base on the page's input
  - `$scope.changeRequestDates` [Ted], This will modify the request base on other functionality
  - `$scope.clearManagerForm()` [Ted], Clear out all the inputs

- `$scope.changeEmployeeView()` [Ted], This will change the user interface depedning if the user is a manager or not.
- `$scope.targetPendingEmployee()`[Ted], This will select a pending employee
- `$scope.init()` [Ted], This will initialize all functions that are needed to run the request page. (This was mostly used for debugging and testing)

- **schedule.js**
  File to control schedule UI functionality. This file was coded by all members of the team.
  - `$scope.autoGenerate ( )` [Jasjot], UI functionality for triggering auto generation of schedule
  - `$scope.publishSchedule ( )` [Amador], marks a schedule as published
  - `$scope.selectPosition ( )` [Beth], verifies pid and title to select position
  - `$scope.boardDragControlListeners ( )` [Beth], This function handles events from an external library that allow dragging the schedule template shifts.
  - `$scope.gapDragControlListeners ( )` [Beth], This function handles events from an external library that allow dragging the spacers in the schedule template.
  - `$scope.selectFromList ( )` [Beth], sets selected id when clicking employee list in schedule
  - `$scope.selectEid ( )` [Beth], selects the employee id
  - `$scope.employeeListHighlight ( )` [Beth], highlights list on the side
  - `$scope.inputStatus ( )` [Beth], validates schedule edit input
  - `$scope.updateChangesMade ( )` [Beth], saves changes made from factory

/src/main/webapp/WEB-INF/js/directives/

Angular directives - custom HTML element used for front end schedule

- **select-on-click.js:** This directive was setup to handle click events on the schedule edit. It utilizes standard jquery events and applies all events to the given element with attribute marked as ‘select-on-click’. It is meant to add to html input elements. This file was coded by Beth.
  - `element.on(‘click’)` [Beth], this click handler selects the entire length of the input so it will be removed when the user presses the next key, this is so they won’t have to backspace the entire string to change a name. When a name is clicked and a key is pressed, they are usually changing it.
  - `element.on(‘keydown’)` [Beth], minor selection quirks addressed.
  - `element.on(‘keyup’)` [Beth], this event is the bulk of what is being checked when the user is typing in the schedule edit. It grabs data from the element about the schedule

template. Each time a key is pressed, the list of employees are checked for a match. When a match is found, it then changes the style class of the element to signal that a valid name was found. It then checks if the employee is current, if they are available for that spot, if they have the required position for that spot, and updates change lists accordingly so the schedule is properly saved. It then sets that employee's id as 'selected' so that all other spots where that name appears are then also highlighted. If the user removes a name, it is put on the list to delete.

/src/main/webapp/WEB-INF/js/factory/

Angular factory - able to be used by controller when injected as a dependency

- **auth-token-interceptor.js:** The purpose of this factory is to inject an authorization token with each request being sent, and store a token from each response received. This file was coded by Beth.
  - `tokenInjector.request ( )` [Beth], This function pulls the auth_token from localStorage and appends it as the 'Authorization' bearer in the header of the request.
  - `tokenInjector.response ( )` [Beth], This function retrieves the auth_token from each response and stores it back in localStorage.
- **login-factory.js:** This factory handles sign in and sign out procedures from both google and local with this application. This file was coded by Beth.
  - `appSignin (gapi )` [Beth], This function handles sign in with the local application after google sign in was verified. It grabs user information from the database.
  - `destructData ( )` [Beth], This function logs the user out of the local application
  - `googleSignIn ( )` [Beth], This function handles google sign in and gathers data about the user from google.
  - `switchUser ( )` [Beth], This function signs the user out of google for this application.
- **request-factory.js:** This factory is for handling user vacation requests. It has not been fully implemented yet, It only contains functionality necessary for sharing across controllers. This file was coded by Beth.
  - `initPending ( )` [Beth], This function initializes pending vacation requests.
- **schedule-factory.js:** This factory contains inner workings of the scheduling. It is responsible for gathering the main schedule template, modifying it with calculations to save computation as scheduling occurs, initializes the week header, tracks the index of the shifts for adding spacers, and balances three lists to decide the state of the schedule. This file was coded by Beth.
  - `initMonday ( )` [Beth], this function finds the most recent monday to use as a starting point to initialize the week header.

  - `initHeader ( )` [Beth], this function uses monday to initialize the date for the other days.
  - `initRequests ( )` [Beth], this function pulls vacation requests for the week for signaling that an employee is not available for a given shift.
  - `initHolidays ( )` [Beth], this function pulls holidays for the week for signaling that a day is marked as a holiday.
  - `initSchedule ( )` [Beth], this function pulls in the schedule template from the database.
  - `saveSchedule ( )` [Beth], this function saves the schedule in the database
  - `deleteSchedule ( )` [Beth], this function deletes the schedule in the database
  - `saveShifts( )` [Beth], this function saves the order of the shift indices.
  - `countDays( )` [Beth], this function counts days that each employee is scheduled for on the given schedule.
  - `countHours( )` [Beth], this function counts hours that each employee is scheduled for on the given schedule.
  - `trackScheduleChange( )` [Beth], this function tracks which list a schedule change belongs in: delete list, or update list. It disallows items from being added to both, and doesn't add changes that were in the original template retrieved from the database.
  - `clearSchedule( )` [Beth], this function clears the schedule by adding all shifts into the delete list.
  - `importWeek( )` [Beth], this function imports the selected week to overwrite the schedule with that data.
- **user-factory.js:** This factory is responsible for managing the employee list for the application. This file was coded by Beth.
  - `initUsers ( )` [Beth], this function pulls all employees from the database.
  - `buildAvailability ( )` [Beth], this function constructs an object of the employees' availability that is convenient to check dates against.
  - `isAvailable ( )` [Beth], this function compares employee availability to a given shift on a given day. It decides if the employee is available or not.
  - `buildPositions ( )` [Beth], this function constructs an object of the employees' positions that is convenient to iterate.

/src/main/webapp/WEB-INF/styles/

CSS styling uses SCSS

- **main.scss:** This is the main style sheet constructed. This file was coded by all members of the team.

/src/main/webapp/WEB-INF/views/

Views are the 6 main pages, each view contains zero or more templates

- **employee.html:** Employee edit view main source for employee view, contains other employee nested templates This file was coded by Beth, Amador, and Jasjot.
- **home.html:** Home view main source for home view, contains other nested templates. This file was coded by Beth and Amador.
- **login.html:** Login view was coded by Beth
- **manage.html:** Manage view, contains other nested templates for manage This file was coded by all members of the team.
- **request.html:** Request view, contains other nested templates for requests This file was coded by This file was coded by Ted and Beth.
- **schedule.html:** Schedule Edit view, contains other nested templates for scheduling This file was coded by Beth, Jasjot, and Ted.

/src/main/webapp/WEB-INF/views/templates/

Templates are sections of HTML included in views. They are prefixed by their corresponding view

- **employee-availability.html** This file was coded by Jasjot and Beth.
- **employee-history.html** This file was coded by Jasjot.
- **employee-info.html** This file was coded by Jasjot.
- **employee-list.html** This file was coded by Beth and Jasjot.
- **employee-permissions.html** This file was coded by Beth.
- **employee-positions.html** This file was coded by Jasjot.
- **footer-menu.html** This file was coded by Beth.
- **header-menu.html** This file was coded by Beth.
- **manage-deliveries.html** This file was coded by Amador.
- **manage-holidays.html** This file was coded by Ted.
- **manage-restrictions.html** This file was coded by Beth.
- **manage-shifts.html** This file was coded by Jasjot.
- **request-history.html** This file was coded by Ted.
- **request-manager-mode.html** This file was coded by Ted and Beth.
- **request-pending.html** This file was coded by Ted.

- **request-submit.html** This file was coded by Ted.
- **schedule-delivery-edit.html** This file was coded by Amador.
- **schedule-edit-options.html** This file was coded by Beth and Jasjot.
- **schedule-edit.html** This file was coded by Beth and Jasjot.
- **schedule-header.html** This file was coded by Beth.
- **schedule-view-options.html** This file was coded by Beth.
- **schedule-view-print.html** This file was coded by Beth.
- **schedule-view.html** This file was coded by Beth and Amador.
- **shift-info.html** This file was coded by Jasjot.
- **shift-list.html** This file was coded by Jasjot.

/src/main/webapp/static/

Static content

- **index.html:** The starting point for the view tree, this contains all other views within it and displays appropriate views based on the URL route. This file was coded by Beth.

/src/main/webapp/static/bower_components

Contains directories for library dependency code installed by bower when it reads bower.json. This folder is not under version control.

/angular: library for the Angular framework
/angular-animate: library for Angular javascript animations
/angular-aria: library for Angular accessibility for web pages
/angular-cookies: library for angular cookies
/angular-local-storage: Angular wrapper for interacting with browser localstorage
/angular-material: library for Angular extensions such as date picker, modals and popups
/angular-messages: library needed for
/angular-mocks/
/angular-resource/
/angular-route/
/angular-sanitize/
/angular-translate/
/angular-translate-loader-static-files/
/angular-underscore-module/
/bootstrap-sass-official/
/font-awesome/
/jquery/

/moment/
/ng-sortable/
/underscore/

/src/main/webapp/static/i18n/

Internationalization (i + 18 letters + n). Used for Spanish language translations

- **locale-en.json:** English definitions for translations. This file was coded by Beth.
- **locale-es.json:** Spanish definitions for translations. This file was coded by Beth.

/src/main/webapp/static/img/

Images

- **favicon.ico:** Browser tab icon
- **user-icon.png:** Image used for profile photo when the user has not set their google account photo.

Database Diagram

Entity Relationship Diagram

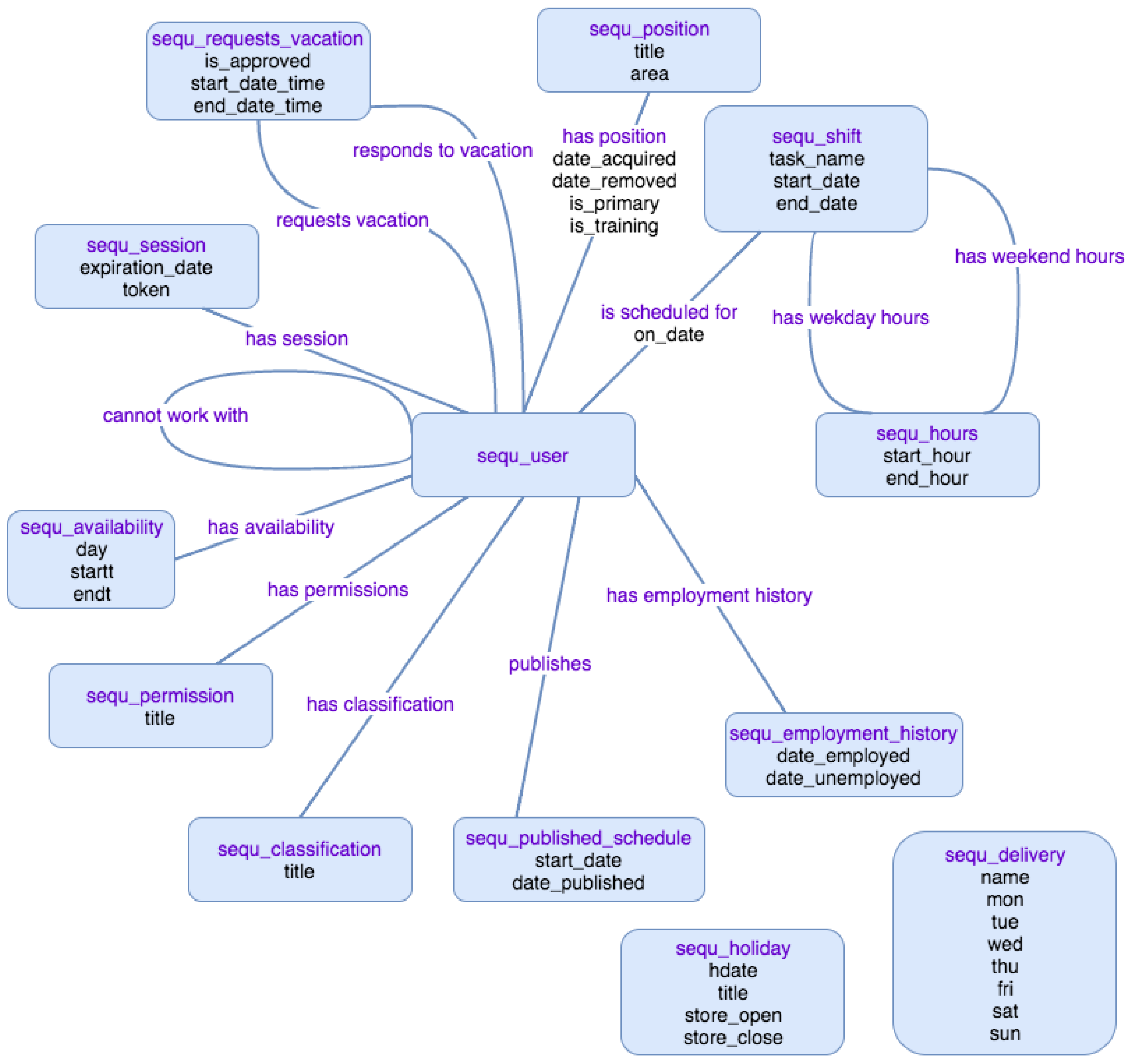


Figure 2

Individual Code Diaries

Bethany Armitage - Senior Seminar B

Week 0-1(03/21 - 04/03)
Arranged meeting with business that product is being used to model scheduling system after to discuss the progress of the application and ask for feedback. They suggested adding a notes field for the employee. I noted that down to work on later. Still translating database. (stuck on procedures with JDBC - only functions exist in PostgreSQL, which must have a return value. Regarding project management overhead, we decided to redesign how to work on the project. We decided to set up the team in partners instead of individuals we we could have a dedicated person to work with. I advised Sunny on the procedure for installing dependencies for the Java back end. This is where we use Maven to install them.

I advised Ted with closing out holiday issue, and design of the UI for the holiday edit in manage store view. Work with AJ on constructing the workflow of weekly scheduled deliveries. The weekly deliveries will be displayed on the schedule for the convenience of the employees working that day. Work with restructuring the http request interceptor to accommodate planned changes for security implementation. Work on finishing some small styling changes. Add lines to the table showing on the printer friendly page view. Add links to the footer for polishing and useful information. It now shows the license, links to the source code on github, and show current application version. It still has the link to translate the application to Spanish, but that option is not fully implemented as of yet.

Week 2 (04/04-04/10)
Solve problem translating database. It turns out there was hard coded value, Plan two team meetings Wednesday is Auto Schedule and Friday is a meeting discussing scrum, final project report jumpstart and questions. Implement Liquibase versioning system by setting up initial changelogs to build database. Continue with Liquibase for other files than base tables, such as procedures, functions, sequences, and views. Connect liquibase to local database so the DBMS reflects the changes.

Week 3 (04/11-04/17)

Put interceptors in java to verify that it will intercept a hard-coded auth token. More setup for interceptor on back end. Force ordering for all async calls so that requests don't overlap and kick the user out when requests will require passing an api token back and forth. Put dev mode logic in so that page refreshes more quickly by skipping long database queries involving nested views. Move 'global' like variables to be stored to application root instead of controller. Use local storage to store the last path the user entered so on refresh, it goes to the previous page instead of application root. Each time the view switches, it updates the last path in local storage as to keep it current.

Week 4 (04/18-04/24)

Mozilla Persona is still implemented, but we skip persona with a default token, and no verification of tokens in place yet, then work on removing Mozilla Persona. Send sha512 hashed password to check against database when logging in. Haven't implemented any password initial setting. Haven't implemented resetting password. Does not verify that the user rightfully owns the email they are claiming to sign in with. Research design of how the implementation of api tokens works. Learn about the difference between security measures available to web applications. Added the notes field for employees as suggested by the business modeling the scheduling system after. This was simply adding a text box to the employee edit UI, a column to the user table, and adding it as a parameter to be sent as the request to update the employee is sent, and to retrieve when all employee information is retrieved. This also included adding the attribute to the employee info database view so that the information was available in the existing query.

Week 5 (04/25 - 05/01)

Create the concept of user permissions. This meant creating a table of permissions and inserting data into the permissions table. Also, each permission needed to be retrieved when the user logged in. This meant adding attributes to existing view and adding data to the join table user-permissions to associate permissions with users so the views would have the attribute. Each Java method that accepts requests had to be classified as requiring specific permissions, and needed to be setup to return if the required permissions were not met. Since this would return an error, I needed to research how to modify the response status to something other than 200. I found that I could set the status directly in the post handler for response interception. If the user tries to access data they shouldn't, the permission is detected and the status 403 unauthorized is added to the model. The model is checked in post handle, and then appends the appropriate response code. Also, users can be classified into more groups than just Employee or Manager now. This required creating and inserting classifications in a classifications table. The 'isManager' column changed to become a foreign key reference to the new classification table. Update the Java User class to include a classification and a CSV list of permission IDs. Rename all columns containing employee_id to user_id.

Update database functions, views and sequences to accommodate the new tables. Embed scope of access within JSON web token (user permissions). Pull the scope out of the token and set the list as an attribute on the request in preHandle interceptor. Then, pull the permissions from the request in the postHandle interceptor in order to embed them in new token.

Successfully retrieve permissions in one back end method, and check them. Update list of permissions to be retrieved as list containing their titles instead of IDs. Continue to rotect more methods with checks that the user has the required permissions to continue. Each back end controller now has a method to parse the permissions that were embedded within the api_token. It converts the CSV list to a List object, and sets the list as a parameter in the endpoint. Each method will then check the list and allow users with permission 'admin' to continue, or the permission specific to the method. For example, if the endpoint "/sequoiagrove/employees" his hit, the user must have permission "manage-employees" or "admin". Start building a factory for schedule to move methods out of main and schedule controller. Factories are better because controllers load and unload upon switching views, whereas factories are consistent. Work more with token passing but not all done, dev mode re enabled, login by sending token instead of credentials almost working. Successful token passing for user initiated login sequence setup initial data, no expiration date checking implemented yet.

Week 6: (05/02-05/08)
Closed issue #26, change "import last week" to be "import selected week". The schedule edit view now has a dropdown list displays the previous 5 weeks, and when the user selects one, it will import that week into the current schedule. It is setup to not save the changes until the user clicks the save button. Worked on some basic cleaning and code formatting where there were unnecessary imports that needed to be removed from some of the Java classes. Start on

another changelog for more db updates to remove the is_manager column from user now that we have several classifications. Do some cleaning to remove unnecessary imports. Simplify error message for login by returning a message instead of a different boolean for each different type of error.

Week 7: (05/09-05/15)
combine employee and user classes to only be a single user class, both classes were redundant because the Employee and User classes both had overlapping attributes. Add an error console to the schedule edit view. If there exists an error (the name is red), then clicking on that space will reveal all existing errors. Currently only checks that the employee has the position for the space they are being schedule for. Plans are in place to additionally check the employee's availability, if they are a current employee, if they have requested vacation that day, and if the day is a holiday or not.

I was really excited when I found a javascript library to implement drag and drop for the UI. I used this library for a few things. One cool feature I was able to implement working on the schedule was spacers to break the schedule up into groups. A well of cloning spacers loads, and they are able to be dragged in between the shifts to push them apart and it creates a space. This works by skipping an index in the shifts. The shifts are saved with their current indices, and when they are pulled into the application to be built, it checks whether they have a skipped index, and if it does, it adds a spacer in that spot. The spacers can be arbitrarily ordered or removed. holidays are also highlighted, employee requests approved are shown as not available on the schedule, add default permissions when adding a new employee. Add liquibase file to save the indices of the shifts in the database. Originally, it was not able to be saved. Because the drag and drop library was implemented, it now allows for arbitrary ordering of positions. Also add spacers so the shifts can be grouped arbitrarily as well.

week 8: (05/16-05/22)
Work on Google signing in and out of the application. Show more detailed errors for problems logging in. It shows if the user is not listed in the database for the application, and it shows an error if they are no longer a current employee. prepare to add visual indication for saved employee, as it currently does not indicate to the user whether the save was successful or not.

Solve the problem where the Google button would not consistently show up. Sometimes it would not render. This was because it was loaded after Angular had rendered the view. This problem was solved by loading the button in app.js so it would render at the right time. Add Factory for user and login. Login factory controls current logged in user and contains sign in and sign out functionality with Google and this application.

week 9: (05/23-05/29)
Fix errors after google login implemented. Give ability to switch user when logging in with google. Cleanup workflows regarding google login. For example, when a user logs in and out repeatedly, keeps app in the state it should be in based on user permissions. Add position checking to the front end of the schedule. When the schedule template is built, in main.js, it iterates the object, and adds an object where the keys are the employee ids, and the values are if they have the given position or not. This attribute is calculated and then added to the schedule template object so it is easier to check later instead of running the computation.

Added the availability is added to the Schedule Template Object in the same way. For each shift in the schedule template, it gets each employee, and finds out if they are available or not for each weekday. This is computationally expensive, and uses more memory, but the tradeoff is worth it for quicker checks as the user is editing the schedule. Also added that it would check if the user is a current employee or not.

week 10: (05/30-06/05)
Work on polishing small things for the look of the application as well as the code. As I come across small errors, I work on fixing them. I removed some old commented out code. Use low level reload when a user without manager privileges signs out of the application and it then switches to a user who is a manager so manager functionality. doesn’t linger and cause requests to be called where the user is unauthorized. Project documentation and report. Add missing screenshots to documentation. Lead managers directly to pending requests tab, as they are more likely navigating to requests to answer them than to submit their own. This was done by adding a catch to switch tabs to pending when the controller for requests page loads. The page will load when it enters the scope. Remove debug statements throughout application. Remove trailing whitespace from a few files. Make connections to factories with local variables that were not updating properly.

Ted Pascua

Seminar B:

Week 1

Team responsibility change.

This week in our team meeting, we talked about improving our work in our application. So we decided that we should work in pairs and specialize in either the backend or frontend of the application. I got partner up with Sunny to work on the backend coding for this quarter. So, I'll be mostly working on the java base functions and queries for the database.

Holidays

I still had to work on finishing the holiday page and holidays interaction with the schedule page.

The holiday interface used to be at the same page with the schedule page. Now, the holiday interface is stored in the manage store page. It got an interface overhaul and it's now easier to modify, add, and remove holidays.The holiday entries will now have a title, date, open hours, and close hours. It needed to be more dynamic because the sequoia grove store can close for half the day, a quarter of a day or close for the whole day.

Once the schedule page is loaded, the page will try to receive holidays from the backend that will have similar dates as the current week. Then based on the holiday entries, we set 7 booleans that will be named after each day of the week and set them true if it's a holiday and false if it isn't.

For example,

(Holiday: PresidentDay, 5/18/201X, Monday).

('mon': 1' 'tue':0, 'wed':0, 'thu':0, 'fri':0, 'sat':0, 'sun':0)

The schedule will grey out one of the days columns if the day is a holiday.

Right now it doesn't grey out specific hours.

Week 2

Database Change.

We are now setting up a new database called PostGresSQL. We had to install a few programs: Postgres, Liquibase, and PGADMIN and anything that can access postgres database. We had to do a few things that were kinda complicated to do. I had to work with Beth to figure out how to setup the server on my computer. We had to do some research on how make the postgres database, initialize the database, and make the application use that new local database. I had to configure my computer account and set up a postgres account so I can do postgres activities. It was a long and complicated task, but we manage to run the postgres server in my computer. Then later, we had to help everybody else set up the postgres database. Now, everybody on the team was able to locally run postgres server.

Autogenerate Schedule

In our meeting on friday we talk more details on designing the auto generate schedule. I was going to work with Sunny on how to implement this. We needed a dynamic way to fill in the schedule with past schedule information. I'm still a bit confused on it though I need to keep asking questions. We also talk more about security. The other teammates were going to work on that.

Holidays and Schedules

The holiday interfaces is nearly done with its interactions with the schedule. We just need to add functions that limit the schedule days and times of employees. After these functions are finished We can check the github issues of holidays marked as done.

Week 3

PostGres

With the database changes, the backend queries had to change because of it. Each of the controller functions had to be rewritten to work with postgres because sqlplus queries did not match with postgres queries. So, in the meantime we had a setup where we can access the database from sleipnir@delphi or in our local machine depending on what branch we use. While waiting for the queries to be changed, updated, and fixed, I researched more about Postgres, Liquibase, and pgadmin. I looked and tried to understand the tools of Pgadmin with Postgres. With Pgadmin, I was able to run and test variety of queries. I was able to make my own table and view. I was able to order them and group them by numbers and type. I also learned that Postgres has a different way of sql commands.

Autogenerate Schedule

We talked and explain more on the design of the auto generated schedule. We needed a way to represent past schedule information to start doing what we need to do. We talked about making a hash tables that will get the each employee in each shifts for one day. We do that for each day for the whole week (monday, tuesday, … sunday. Then store them inside some kind of array or list. Then we find out the amount of times there are replicated data (ignoring the dates). Then we make a new hash table that is the same thing but with no duplicates. But, instead of employee being its only value, it will take in account for the amount of times the employee worked on the same shift and day. Then we have to trim it by employees who can not work together, employees who are not available on that day, We also need to add priority to the best type of workers who can work on that shift for that day.

I am starting to get it now, and I am planning on coding how it works with Sunny.

Holidays

Most of the Holiday interfaces is now done. It works well and It will grey out cells on the schedule page if it is a holiday. We can now check the github issues of holiday interface off.

Week 4

PostGres

This week, I spent a lot more time on working with postgres and liquibase. I learn more from Beth how to work with Liquibase. With liquibase I learn how to run and initialize tables using liquibase files to make tables and views. Liquibase helps us set up databases on different computers. We all have our own local databases and github branches to work on. We might not have the tables of the other people in our group. Even with branches in Github, it doesn't give us the tables stored in our local machines. Instead, we make liquibase do that for us. We write down the tables inside the liquibase files. So, when we run our application, liquibase will read that file and make the table inside our database respectively.

With postgres and liquibase, we don't have to worry that much about messing each other code with the database mess up because it's not on one server.
Also I read a few pages on PostGresSQL.

Week 5

Autogenerate Schedule

We started on making a hashmap that gets each employee in each shifts for one day. We made a hashmaps with the shift being the key and the employee being the value. Then we make an array of size 7 that represents the days. For each slot we make a hashtable getting information for each respected day. But we can simplify it. Instead, we made a 3 dimensional hashmap to represent that. In postgres, I made a query that simplifies getting that information so it's easier to get what we need.

```
select day, shift_id, employee_id, count(*)
from employee_shift_view
group by day, shift_id, employee_id
order by day, shift_id, employee_id
```

This will get for a list of employee for each shift for each day. This will be able count the number of times for the same entries ignoring the dates. This list was stored as a dayShiftEmployee object. Then it was converted into a 3 dimensional hashmap. The days and shift acted like keys. The end values were the employee and the amount of times he or she worked. We were able to present schedule in a neat fashion.
For example.

```
Monday
    Shift 1
        Employee 001 Worked 2
        Employee 002 Worked 3
    Shift 2
        Employee 003 Worked 4...
Tuesday
    Shift 2
….
.....
….
```

The 3 dimensional hashmap is part of the Generator.java object which will have functions that modify the information inside the hashmap.

<u>Week 6</u>

<u>Autogenerate Schedule</u>

We added a few changes to the Generator.java hashmaps. The First element in the hashmap used to be a Integer key. We changed it to a String key to represent the days better. We also fixed some bugs and optimize the code to perform better and look good. When adding into the 3 dimensional hashmap from the database information. Procedure were made to prevent bugs. When adding a an object inside the 3d hashmap, the previous keys must exist first.

For example

[Day, [Shift, Employe]]]

Key   Key

If the previous keys don't exist it will cause a segmentation fault. So when adding the dayShiftEmployee object into the hashmap, it makes sure to add the keys in first.

<u>Week 7</u>

<u>Autogenerate Schedule</u>

The generator.java object has built in functions that has calls the database to get Information from the database. We needed to add function that will start trimming down the generator. We added a List of Employees entries that matches with the past schedules. We added a List of Shifts entries that matches with the past schedules. We added a List of Request entries that matches with the past schedules and matches with the list of Employees.

<u>Week 8</u>

<u>Autogenerate Schedule</u>

It was a bit hard to trim because we had to convert the hours to something comparable to other List of other objects.I started added the trimming functions for the schedule. They are in prototype mode and some parts of it are in pseudo code.

Week 9

Documentation

For this week I mostly worked on the documentation. I tried to work on fixing minor Issues for the application.

Bug Fixes and Improvement.

There was some bugs that may not look good on presentation and we needed to scan for bugs that will mess up the code. I was given a todo list in the coding to make the code a bit more functional and look better.

Week 10

Documentation

For this week I mostly worked on documentation for the project. After the documentation we are going to find out bugs and fix our code so it looks neat on our presentation on monday.

Amador Silva - Senior Seminar B

Code diary week 1:
Delivery issue. Made functions to add deliveries $scope.addDelivery.
Function to delete deliveries $scope.deleteDelivery. Round trip process was used. Dummies boolean were set in the database that reflected the delivery days to the front end. Front end js function got data from the front end when a new company was added for instance, send that data through an http request to the back-end. JSON and GSON had to be used for proper conversion of he data and/or objects from the front end to the backend. The backend then sends the sql updates to the database so the database tables can get updated.

Code diary week 2:
Week 2 was not intensive in code, however time was used to install tools that would facilitate the coding and development process. tmux is a software application that can be used to multiplex several virtual consoles allowing a user to access multiple separate terminal sessions inside a single terminal window. It is useful for dealing with multiple programs from a command line interface and for separating programs from the Unix shell that started the program.
As the project contains more than two dozens files and working on different files became a challenge with having multiple windows open, installing tmux will alleviate a lot of the project logistics.

Code diary week 3:
With the move from Sleipnir's Oracle database and plans to eventually move the project to Amazon Web Services, the choice was made to use PostqreSQL as a database. For this project, each group member would have a version of the database on their own local machine. With the help of tutorials and other group members the installation process went fairly well. The database being an important part of the project, the next step was to write test queries to ensure that the correct information was getting retrieved every time a query was sent from the back end to insert, update, delete or pull.

Code diary week 4:

Security. This week we started looking at handling security through the use of API tokens. I worked on learning what API tokens are, the process through which a token is sent from the front through interceptors that handle transmitting the token to the back end. The token is then sent back with the information requested, in the meantime the token is changed while it is sent back to the front end.
I also set up a session table in the database. That table is indeed needed to store the token, its expiration date and the employee_id indicative of the entity that made the information request.
I also wrote some queries that would help update the token and the expiration date, queries to delete the token.

Code diary week 5:
Work on security (continued), this week, the progress on the issue of API tokens slowed down because of concerns that our code doesn't accomplish what is needed. The main concern was that we were able to intercept the token and “steal” it and make requests for information. In other words, we were able to break our own security, or so we thought. After doing more research, we found out that the API token passing that we implemented was actually performed right. The ability to capture the token was only made possible because SSL encryption hasn't been implemented.

Code diary week 6:
This has primarily been a week of debugging. This is because tables have been modified in the database. The Employee table was turned into the User table. The permission, the user-permission as well as the type and session tables were added to the database. This in turn created many conflicts with the code we already had in place. In the controller directory, the file *Authentication.java* file, when a new user object was instantiated, an *ArrayList<String>* had to be passed as a argument to replace a String argument. This is because the permissions table was added permissions for a specific user was passed to the User constructor as an *ArrayList<String>.*
In the model, the *User.java* file was modified. A User constructor was added to reflect the changes made to the database. As a matter of fact, the second User Constructor was added to allow for User object to be made that contain both the data that was formerly contained in the original database. The proper setters and getters had to be set up as well.

The file *SuperUserRowMapper.java* was added to the model, retrieves row from the database from the new columns of the user table, it also parses the ArrayList<String> to retrieve important information that contains the positions, availability, permissions and history.
Once these additions were made to the code, there were multiple errors because of conflicts between the old attributes in our code and the updates that were brought because of the different updates. Going over and debugging those errors took hours of work once our code and database were updated.

<u>Code diary week 7</u>: This week, I looked into the concept of Javascript factories. In javascript, a factory has constructors which create objects without the keyword new. Using factories would, in fact lead to better code optimization. In the case of the delivery issue for instance, using a factory in conjunction with a delivery Javascript file would serve to reduce the number of constructor calls in the code. For the delivery issue, the steps to implement a factory would consist of initializing an empty service variable and an empty array. The 'main.js' file would have to be initialized with delivery objects as well. The function '$scope.getDeliveries' would need to get moved into the factory that is to be created. In summary, the factory would automate the instantiation of deliveries objects.

<u>Code diary week 8</u>: This week the issue of implementing a bootstrap badge showing the number of requests pending in the header menu was started. The process involves getting the total pending request from the *getpendingrequests()* function in the *request.js* file. The *main.js* file was modified to hold the necessary data that would enable the the pending requests to be read properly. The html template file had to be modified also in order to show the correct data on the request page.

<u>Code diary week 9</u>: Week 9 consisted of debugging issues related to the requests page. After passing the value for the pending requests to the proper javascript and html files, the default value of zero is still displayed in the requests tab even when the value for the pending request is superior to zero. The debugging process included using the browser console to obtain the value for pending requests in the *request.js file.* At first, the correct values for the requests pending were obtained in the browser console . Then an error occurred. The error pointed to the bower package manager that our project uses to install, track and manage packages and external dependencies. The bower components are external to our project however and shouldn't be

subject to errors since they are imported packages that diddn't give any errors previously. Reinstalling bower and its components did not get rid of the error. After a multitude of other modifications and re-installation, the error persisted. Switching browsers from Firefox to Google Chrome proved to satisfactorily get rid of the error. At this time, there is no understandable find that would justify why the browser used to test the front end of the application would generate errors related to a package manager used install dependencies, especially when testing had been conducted on my end using Firefox for more than five months without any errors related to the bower manager. Research into that particular issue continues.

Code diary week 10: This week was essentially dedicated to pushing the last changes made to the code. In this case, the last fixes made to the requests issue were pushed onto Github. In order to avoid any last minute bug, it was decided that this week would the last week to make any changes to the code. This would give just enough time to resolve any merge conflicts or otherwise technical issue. A majority of the time was also dedicated to finishing up on the code documentation. Code documentation has been a group effort, I have contributed in miscellaneous parts of the report. This week will end on preparing for the group presentation of the actual project.

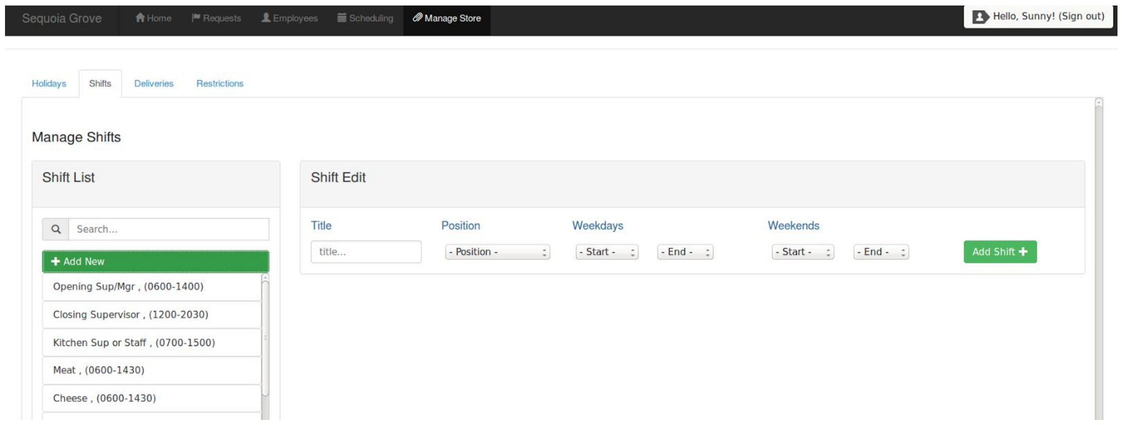
Jasjot Sumal - Senior Seminar B

Week 0: March 20th, 2016 - March 27th, 2016 (Spring Break)

With the time I had available this week, I moved the shift edit view from the Scheduling page to the Manage Store page, under the Shifts tab. Since the shifts are no longer available for the user to click on, a scrollable list was added to the Shifts tab so the user has a means of selecting the shift the would like to edit. The employee list in the Employees page was used as a template for this list, so the code for these lists is very similar, meaning the function similarly and have the same look and feel. This was preferred since it creates a consistent user experience. The shift edit information is similar to the UI created earlier for the scheduling tab, but is placed in a border and header similar to the view seen in the Employees page. Also just like in the Employee list, the shift list has an 'add new' button for adding a new shift, and the shift currently being modified is highlighted in blue.

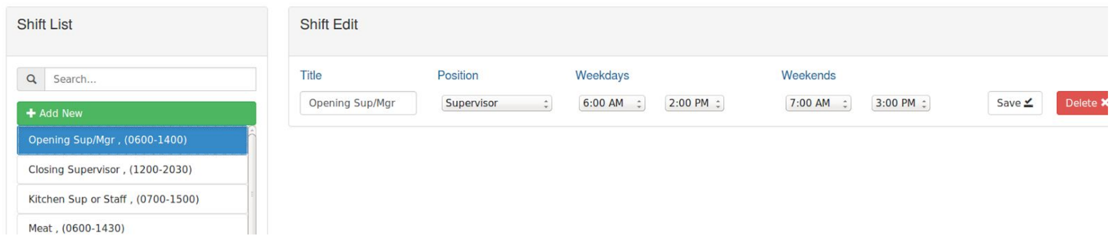
Corresponding Javascript functions and variables were copied into the appropriate controller for the Manage Store view, and the shift edit Java functionality was copied from the Schedule controller to the Manage Store controller in the back-end. Once this was done, the views under the Manage Store tab were working as needed after some tweaking.

Final Views:



Opening Sup/Mgr , (0600-1400)
Closing Supervisor , (1200-2030)
Kitchen Sup or Staff , (0700-1500)
Meat , (0600-1430)
Cheese , (0600-1430)

[Adding New Shift]



[Editing Shift]

Some finishing touches still need to be added to make sure all the buttons fit into the view for any user while editing or deleting the shift, however this is enough for a basic user experience.

Week 1: March 28th, 2016 - April 2nd, 2016

Much of this week was spent determining how to return an error state manually from the back-end. This was desired so the front-end could accurately display an error message or successful save message to the user. After unsuccessfully researching different ways to change the return status, a StackOverflow page was found describing a method for returning a 500 'Internal Server Error' by throwing an exception. Once this was coded into the back-end, an error status could finally be returned for the validation checks being performed.

The front-end could now accurately respond to error or success status sent from the back-end, so I then started on coding basic success or failure notifications for the user. After some experimenting with Bootstrap classes, notifications for successful or failed saves that looked decent were created and made to show only when the corresponding state was returned from the back-end.

With these notifications functioning, this concluded my immediate involvement with creating a shift edit page. Some minor styling and real-time help for helping users through errors still needs to be done, however this will be saved for a later time. I will now move on to planning, implementing, and leading the development of AI for automatically generating the weekly schedules.

Week 2: April 3rd, 2016 - April 9th, 2016

This week was spent planning out the AI for auto-generating the schedule. Ted Pascua and I were assigned the task of planning and implementing this AI by the team. With the other issues we were working on taken care of, work could now begin on auto-generating the schedule.

Planning started by determining the requirements of the AI. The task of this AI is to automatically fill in the current week's schedule with an acceptable employee for each shift. This means the AI must make sure each employee scheduled is available during the shift time, has not requested the day off, is trained in the position, is able to work with other employees that have shifts during the same time, and, ideally, is the person that usually works the shift.

This means the AI requires the corresponding information from the database. Employee availabilities and requests off taken during the week are required. Also needed are the positions each employee is trained in, so they can be assigned to shifts in case all employees who worked the shift in the history are unavailable.Finally, a list of employees are needed for each shift on each day. This list will contain the employee IDs of each employee that worked the given shift slot for the past X weeks, where X is a number provided by the user.

These lists of employee id's for each shift slot will be the essential data used to construct the schedule; they will be turned into hashmaps for each slot, where the key is the employee ID and the value is the number times that employee worked that slot in the past X weeks. Next employees that are able to work the shift slot (know the position) will be added to each slot's hashmap with a value of 0. Employees that are not available for a slot, or have requested the slot's day off, will be removed from the corresponding hashmap. Finally the schedule will be filled in based on the employees that have the greatest value in the hashmap for each slot.

Week 3: April 10th, 2016 - April 16th, 2016

After some consideration and meeting with the team, we found our initial design will experience some problems. Firstly, this could lead to the same employee being scheduled twice or more in a day. It could also lead to over-scheduling some employees while under-scheduling others. To solve this, not only must each employee's min/max hours must be check each time they are scheduled, but the AI must attempt to give each employee roughly the same amount of hours. The 'cannot work with' relationship among employees also has not been considered yet, which means before each employee is scheduled, they must be checked against other employees already scheduled during the same time to ensure they can work together. Finally, scheduling employees in this way could lead to the AI getting 'stuck', meaning no employee may be available for some slots.

These problems can be solved by trimming and reprioritizing hashmaps, but the hashmap must then be able to store priority and trimmed employees must be saved somewhere in case back-tracking is needed if the AI gets stuck. However, we currently do not know how we would implement a back-tracking system, so slots with no employees available may instead simply be skipped. We are currently still researching how these issues will be best addressed.

Week 4: April 17th, 2016 - April 23rd, 2016

After discussing with the team possible implementations to address the above issues, and amount of complexity each solution we were considering would introduce, we decided to meet with Dr. Albert Cruz as a team for assistance in solving the problem.

His first suggestion was to use a genetic algorithm to solve the problem. This would essentially be initializing a schedule with random names, and then checking the fitness of that schedule with a fitness function. All the checks described above would occur in this fitness function. This includes what we've termed the "simple" checks, which are making sure the employee is available, knows the position, does not have an approved request for that day, and how often they have been scheduled for that shift slot in the past. The function would then perform the "complex" checks, which are determining how many employees scheduled during the same time a single employee that employee cannot work with, making sure min/max hours are not violated for each employee, assuring employees are not scheduled twice in a day, and assuring employee are scheduled for about equal hours. The function evaluates these conditions and returns a number between 1 and 0 indicating the fitness of schedule.

Next the genetic algorithm would randomly change some scheduled employees if the fitness does not meet a specific threshold, and the fitness function is run on the schedule again. This method continues until a schedule with an acceptable fitness is found.

We felt this method was too undirected for our needs, since it would likely take an unnecessarily long time to solve once a schedule with just below acceptable fitness was reached. A form of A* search was also suggested, this essentially resulted in what were already considering, but gave us some insight as to how to store information in a way that would allow employees to easily be prioritized as needed, and also how history could be handled and taken care of. Instead of storing hashmaps for each slot, employee information would be stored, each with a hashmap of all shift IDs, with values set to that employee's fitness for each shift.

Week 5: April 24th, 2016 - April 30th, 2016

After considering our design options for the AI, we decided to start coding the AI as originally designed and address complexity issues as we came across them. Ted decided to take the task of acquiring the necessary information from the database. At this point I had to wait for Ted to acquire data from the database. In this time, I quickly coded a front end user interface to test the auto-generating function. This UI was just a button that sent hard-coded parameters to the back end for testing, a menu to change these parameters will be designed and coded in later once the AI is complete. Among the hard coded parameters is the number of weeks to reach back for the shift history, which is set to 6 weeks.

Week 6: May 1st, 2016 - May 7th, 2016

Both Ted and I realized acquiring the necessary information from the database was a bigger task than we both imagined, so I have now started helping with acquiring information from the database. I have also started helping construct the Generator java class, which handle the bulk of the work done to build the schedule, and the ScheduleGeneratorController java file, which implements the Generator class. While Ted works on acquiring the requests off, I will be acquiring the shift information from the database.

Also completed this week was the migration of the database from the school's server, Delphi, to our local machines by Beth. Liquibase is being used to sync changes made across our machines by tracking change logs. We must manually create the change logs, so it is not the best system, however it allows us to sync our database while keeping development data separate. To implement this changes, I installed Liquibase and PGAdmin on my machine this week, then pulled the latest data and changes from GitHub to fill the database.

Week 7: May 8th, 2016 - May 14th, 2016

After some formatting changes with the date strings being passed, I was able to successfully pull shifts from the database. This task also required adding a function to the database to pull shifts in the correct timeframe, and this in turn required learning how to use Liquibase change logs. However once this was done, shifts were being pulled for the current week.

Week 8: May 15th, 2016 - May 21st, 2016

At this point acquiring requests was not done yet, so I decided to assist with this as well. However before starting this, I created print functions for all data objects in the generator so I could check data was being pulled correctly. Once this was done, I saw the employee information was not acquiring availabilities correctly, so I fixed this, then moved on to making sure requests were being acquired from the database correctly.

Week 9: May 22nd, 2016 - May 28th, 2016

At this point, although the queries were mostly fixed, we were still getting other obscure exceptions and errors being thrown while trying to fill the 3D hashmap object. These were fixed once it was realized there were some more errors in the queries that needed to be fixed, and also that some strings were coming back empty from the database so they needed to be handled as well. After sorting through these problems, the fillGenerator function was correctly filling the 3D hashmap.

Once this was done, we were ready to move on to the “trimming” step, which is removing unavailable employee or employees who have approved requests for days off. This step also involves filling in available employees who do not appear in the shift history, but know the position. However, we realized we could not perform these steps, because all the dates and times were stored as strings so they could not be compared. So I also started work on converting these to comaparable formats.

Week 10: May 29th, 2016 - June 4th, 2016

This week was spent converting the Duration class into number formats that can be compared, then implementing the Duration in in places where it should have been implemented earlier. It was necessary to implement the new Duration nearly everywhere it could be used, not just the Generator class, since the Generator class relies on many of classes to function. A custom date class was also designed since the Java date class was not functioning as intended. This custom date class was implemented in Duration. After making these changes, dates and times were in a format that can be compared.

Although the Generator class is not completely finished yet due to time constraints and the unforeseen amount of work involved in the AI, work will continue on the java class and the AI until it is complete and automatically generating a schedule.