COSC 499 - Capstone

Manpower Scheduling Board

Horizon Electric Inc

Adam Fipke
Jordan Roberts
Eddy Zhang
Edwin Zhou
Yuan Zhu

August 11, 2023

Table of Contents

Identification

Name of project

Manpower Scheduling Board

Team Members

- **Adam Fipke**
- **Jordan Roberts**
- **Eddy Zhang**
- **Edwin Zhou**
- **Yuan Zhu**

Stakeholders

- **Horizon Electric Inc. (Client)**
    - **Dave Clark (Contact person)**
- **Project team**
- **Horizon Electric Inc. employees**
- **University of British Columbia Okanagan (Contract provider)**
- **Clients of Horizon Electric**

Project Purpose and Problem Statement

The purpose of this project is to develop a modern and reliable system for Horizon Electric Inc., a local Electrical Contractor employing 100+ employees, to maintain and provide outlook for future manpower requirements. Currently, the company is using magnets and whiteboards to manage their manpower needs, which can be inefficient and unreliable.

Project Objectives and Related Success Criteria

Detailed manpower scheduling is essential in the construction industry to maintain productivity on all projects. Our project aims to modernize their current system by providing an intuitive graphical interface for Horizon's managers to reliably schedule workers and have automated backups and secure data storage to provide scheduling history and ensure data integrity. This will help Horizon Electric Inc. to effectively forecast their manpower requirements, avoid inefficiencies in staffing, and ultimately improve their operations and profitability.

If looking for the scope and charter document, please see the Scope And Charter.pdf

Requirements

Disclaimer: The strongness of the word "will" is equal to the strongness of the word "must." Treat them as synonymous.

High Level Requirements

- Knowledge of the Linux, Apache, MySQL, PHP (LAMP) web programming stack. ✓
- Experience in designing graphical user interfaces to ensure that the system's operation is smooth and reliable. ✓
- Experience with version control systems such as Git. ✓

Non-functional Requirements

- The system must be constructed with the possibility of having multiple accounts in mind. ✓
- The website will be accessible on the following desktop browsers: Google Chrome, Edge ✓
- The website must be visually similar and function the same across browsers. ✓

- For usability goals in the website's UI, it must prioritise learnability and memorability, meaning that it will not take a large effort to re-learn. We can test this by having the construction managers try to navigate the UI for the first time. ✓
- The websites UI must use appropriate use of colours to identify and indicate important areas. ✓
- All frequent input fields must have both client-side and server-side validation, where applicable. ✓
- ~~Data sent from the client to the server will use HTTPS for encryption.~~
- The software/website must initially load within 5 seconds. ✓
- The system must correctly sum and display the counts of the employees. ✓
- Developers will regularly communicate with each other via Discord and messages must be viewed at least once a day. ✓
- The web server must have a "time-last-changed" variable that the clients will query before pulling all of the information from the web server, reducing web traffic✓
- All open views of the website must be periodically updated every X seconds, displaying any recent changes ✓
- The system must be able to function and run efficiently (I.e., with at most a 0.5s added delay) with up to 3 active clients at one time. ✓
- The system will use a database to hold jobs, their employees, and their history. ✓
    - This must run periodic, daily backups, and hold these backups for up to 30 days. ✓
- The following histories must be saved:
    - What jobs an employee has worked and for how many days ✓
        - Saved on database
        - Incremented at midnight
        - Can be used for visualisations
    - ~~The expected employee counts for each job and the actual employee counts~~ ~ not needed, we calculated it from our *assignments* table which keeps track of who was assigned where and for how long
        - ~~Actual employee counts will be saved over time. Since employees are not always assigned to one job for the entire month, it must be saved as a fraction~~
        - ~~Saved daily at midnight on database~~
        - Can be used for visualisations
    - ~~The history of user actions on the schedule~~ X
        - ~~Saved on client~~
        - ~~Used for undo~~
            - This was not implemented due to the exponential amount of complexity that it would introduce into the project. In the interest of delivering a complete program to the client, the opportunity cost was too great.
            - Basic text undo will still be available through the default browser undo functionality.
        - ~~Updated after every action~~
- Employees are able to work multiple jobs ~~(not more than one at a time),~~ and this must be taken into account when building a history of employees. ✓
- ~~The projector view will have to take in consideration of the colour of the backdrop and must be built in a way that all text will be readable on a 4k projector~~ The presenter view be visually presentable on the client's 16:9 T.V. with a resolution of 3840*2160 ✓

New Non-Functional Requirements

- The system must be responsive in that it will notify the user when changes have been successful or errors ✓
- The system must give the user emphasised warnings for permanent or not easily reversible actions ✓
- The system's server must use an SQL account with minimal privileges (INSERT, UPDATE, DELETE, SELECT, any other required for the system to function) wherever possible ✓

Functional Requirements

- Managers will be able to view a history of previous jobs and when each employee was assigned to the job. ✓
- The system must ensure that recent changes are reflected in all open views of the webpage ✓
- Managers must be able to:
    - Easily allocate and move employees from job to job. Allocate and move by 1-click mouse move. Click the employee who the manager wants to move, and release the mouse at where the employee should be assigned to. This will auto-update the sum of employees for that job, and show a total of all allocated employees at the bottom of the screen. ✓
        - Foremen can be allocated to multiple jobs but must only be counted once in the employee counts. ✓
        - 12 months must be displayed at one time, starting at the current month. ✓
            - Upon the current real month changing, the entire calendar will shift and will stop displaying the last month and will display a new month (rolling calendar). The old month will be archived ✓
    - Edit employee information. Click on the employee's name, then a window pop up to allow manager to edit the details of the employee, click save before submitting the new details. ✓
    - Create new jobs. Click create new jobs button, then a window pops up to allow manager to enter the details of the new job, click save before submitting. ✓
    - Archive Jobs. The user must be able to indicate to archive a job, then the job and its related information will be archived. This essentially performs a "soft delete", where the job is made hidden from most pages but the data of who worked on the job is still kept. ✓
    - Edit job details. Right-click the job that needs to be edited, choose the "edit" option, a window pops up allowing the enter new/update old information. Click save before updating. ✓
    - Edit the number of required employees for a job in a given month, which the system must show a sum of at the bottom of the page ✓
        - The schedule will have indicators on where the expected start and end month is, but must allow users to change the employee count for months after the end month in case projects can go longer than expected ✓
    - ~~The system must allow users to undo previous actions.~~ X Not worth the opportunity cost
- For a job's employee requirements for a month, the system must highlight when there an inadequate, an adequate, and when there is a surplus amount of employees by displaying different colors in the portal. ✓

New Functional Requirements

- The system must be able to archive and unarchive employees, effectively performing a "soft delete" where the employee is no longer visible on most screens (still needing somewhere to see them so they can be unarchived) except the history of what jobs the employee has worked on in the past is retained. ✓
- All visible and changeable information in the old system (the whiteboard) must be similarly visible and changeable in the new system including: jobs and their basic info (titles, project manager, address), employees and their basic info (names, roles, notes), where employees are assigned (a specific job, inactive, in school, or just unassigned), the next 3 months of projections and the sum of these projections for each month (sum does not need to be changeable), and employee sums for each job and a total across all jobs (does not need to be changeable) ✓

List of Technical Requirements

- The website will be developed using HTML, CSS, JS, PHP, and MySQL as the primary programming languages. ✓
- The developers will validate user login with frontend and backend validation and encrypt passwords via hash and salt before storing them in a database. ~ system hashes but does not salt the passwords.
- The system must be able to translate and upload the employees in their current excel sheet (exported to a csv) into the database ✓
- The website is hosted on a local server that does not have access to the internet, therefore no web-based APIs on the server may be used. ✓

List of User Requirements

- Managers will be able to view a history of previous jobs and when each employee was assigned to the job. ✓
- The system must ensure that recent changes are reflected in all open views of the webpage ✓
- Managers must be able to:
    - Easily allocate and move employees from job to job. Allocate and move by 1-click mouse move. Click the employee who the manager wants to move, and release the mouse at where the employee should be assigned to. This will auto-update the sum of employees for that job, and show a total of all allocated employees at the bottom of the screen. ✓
        - Foremen can be allocated to multiple jobs but must only be counted once in the employee counts. ✓
        - 12 months must be displayed at one time, starting at the current month. ✓
            - Upon the current real month changing, the entire calendar will shift and will stop displaying the last month and will display a new month (rolling calendar). The old month will be archived ✓
    - Edit employee information. Click on the employee's name, then a window pop up to allow manager to edit the details of the employee, click save before submitting the new details. ✓
    - Create new jobs. Click create new jobs button, then a window pops up to allow manager to enter the details of the new job, click save before submitting. ✓
    - Archive Jobs. The user must be able to indicate to archive a job, then the job and its related information will be archived. This essentially performs a "soft delete", where the job is made hidden from most pages but the data of who worked on the job is still kept. ✓
    - Edit job details. Right-click the job that needs to be edited, choose the "edit" option, a window pops up allowing the enter new/update old information. Click save before updating. ✓
    - Edit the number of required employees for a job in a given month, which the system must show a sum of at the bottom of the page ✓
        - The schedule will have indicators on where the expected start and end month is, but must allow users to change the employee count for months after the end month in case projects can go longer than expected ✓
- ~~The system must allow users to undo previous actions.~~ X Not worth the opportunity cost
- For a job's employee requirements for a month, the system must highlight when there an inadequate, an adequate, and when there is a surplus amount of employees by displaying different colors in the portal ✓
- Any employee should be able to be duplicated ✓
    - Duplicate Employees should appear separate from their regular counterparts (grey-ed out) ✓
    - Duplicate employees will not be counted in the employee counts and totals ✓
    - Assigning a duplicate employee will still count as the employee being assigned ✓

New User Requirements

- The job view must implement a method for the user to custom sort the list of jobs (even in non-standard orders) ✓
- The system must be able to archive and unarchive employees, effectively performing a “soft delete” where the employee is no longer visible on most screens (still needing somewhere to see them so they can be unarchived) except the history of what jobs the employee has worked on in the past is retained. ✓
- All visible and changeable information in the old system (the whiteboard) must be similarly visible and changeable in the new system including: jobs and their basic info (titles, project manager, address), employees and their basic info (names, roles, notes), where employees are assigned (a specific job, inactive, in school, or just unassigned), the next 3 months of projections and the sum of these projections for each month (sum does not need to be changeable), and employee sums for each job and a total across all jobs (does not need to be changeable) ✓

Terminology

- Projection/Outlook number: numbers that construction manager predicts how many manpower they need for current/future month. The projection number for the next coming 12 months are on the right side of each job, in Jobs page.
- Construction manager: this scheduler’s users, the client
- Admin manager: the scheduler user who have access to admin-functions, including create new account.

User Groups

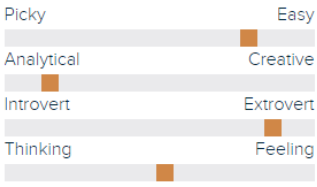
Persona 1 – Primary User

Gregory Thompson - Construction Manager

"Technology hates me."

Age: 50
Education: Bachelor's degree in Civil Engineering
Family: Married with two kids
Location: Kelowna, BC

Personality



Excellent Communicator | Problem Solver | Technologically-unsavvy | Worsening-Eyesight | Detail-Oriented | Resilient

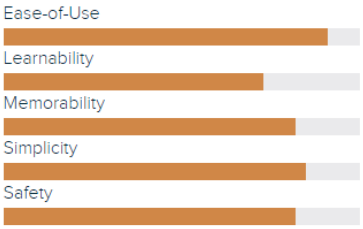
Goals

- Complete all projects within a set timeline and budget, while maintaining quality.
- Be able to view at a glance what employees are working at which jobs, see what jobs need more or less employees and assign them accordingly.
- Eliminate tedious, manual updates of counts in their current schedule board, saving time and improving efficiency
- Create better-informed future decisions by being able to view charts and graphs based on the history of jobs and employees.
- Enhance project coordination by identifying critical path activities and potential scheduling risks in advance.

Tasks and Responsibilities

- Oversee projects and schedules to ensure they are completed on time, within budget, and meet all specifications.
- Manage employees so they are all working to the best of their ability while maintaining their safety.
- Collaborate with his fellow-construction managers within the office to assign employees to jobs in a way that satisfies the manpower and experience-level requirements for the job.
- Communicate with architects, engineers, contractors, and other stakeholders to ensure the projects are done to specifications.

Values



Bio

Gregory is a hard-working, busy manager, who tries to take great care of his employees while also ensuring all projects are done on time. He typically tries to stay away from technology, as he finds himself frequently getting frustrated by overly-complex systems and their hard-to-navigate UI or inability to tolerate his mistakes. Additionally, it can take him some time for him to learn new things, and he finds himself only using a few features of most apps. When Gregory is not at work, he spends his time on the couch watching shows and spending time with his family.

Persona 2 – Secondary User

Alex Mitchell - IT

Age: 27
Family: Married
Education: Bachelor's degree in Computer Science
Location: Kelowna, BC
Character: The IT guy

Bio

Alex spends most of his time carefully listening and helping clients fulfill their needs. He works independently as a contractor to help business with a variety of technical problems. He frequently finds himself frustrated when working on projects that have little documentation, as he finds it very hard to continue where people have left off. In his free time, he enjoys gardening and watching animals.

Motivation

Learnability
Safety
Ease of Use
Function
Simplicity

Major Responsibilities and Tasks

- Maintaining and troubleshooting the organization's IT infrastructure, including servers, networks, and software systems.
- Managing user accounts, permissions, and access controls.
- Ensuring data security and implementing backup and disaster recovery strategies.
- Providing technical support to end-users, addressing hardware and software issues.
- Researching and recommending new technologies and system upgrades to enhance efficiency and productivity.

Goals

- To have a functional way to access and restore databases and view diagnostic data.
- Upgrade the organization's network infrastructure to improve data transfer speeds and enhance overall system performance.
- To have accessible and readable documentation to understand how a system works so he can more efficiently and effectively perform maintenance.

Personality

Introvert — Extrovert
Analytical — Creative
Loyal — Fickle
Passive — Active

Technical | Professional | Dependable

"I can fix that"

Technology

IT & Internet
Social Media
Financial Management
Social Networking

Technical Dependencies

- Docker (Version 4.20) (for running server and database)
    - In dockerfiles, it uses:
        - Apache (version 7.4.3)
        - PHP (latest version from jitesoft/phpunit)
            - Comes with xdebug
            - Runs on alpine linux
        - Latest versions of pdo, and pdo_mysql images
        - Latest version of the ImageMagic library
        - Latest version of mysql image
        - PhpMyAdmin version 5.0.1
- Git (for installing the project)
- Built for Edge (Version 115) and Chrome (Version 115) , also implemented some compatibility so it also probably works on Safari and Firefox

Important Implementation Notes & Global Functions

- All functions that require user input to be put into SQL statements sanitise their input with prepared statements

- ***ajax_search()*** is a very important function that essentially does an asynchronous refresh of the page, which generates all relevant info (employee info, job info, etc.). There is an *ajax_search()* for the job page and employee page, and the history page has a *ajax_search_history()* version as well. This function also gets called when most things are changed (moving an employee, editing a job/employee, searching, etc.).

- ***global_functions.js*** has some very commonly used functions, *generate_overlay()* and *closeOverlay()* for managing the overlays, as well as *send_message()* for showing the little stackable pop-up message on the bottom-left.

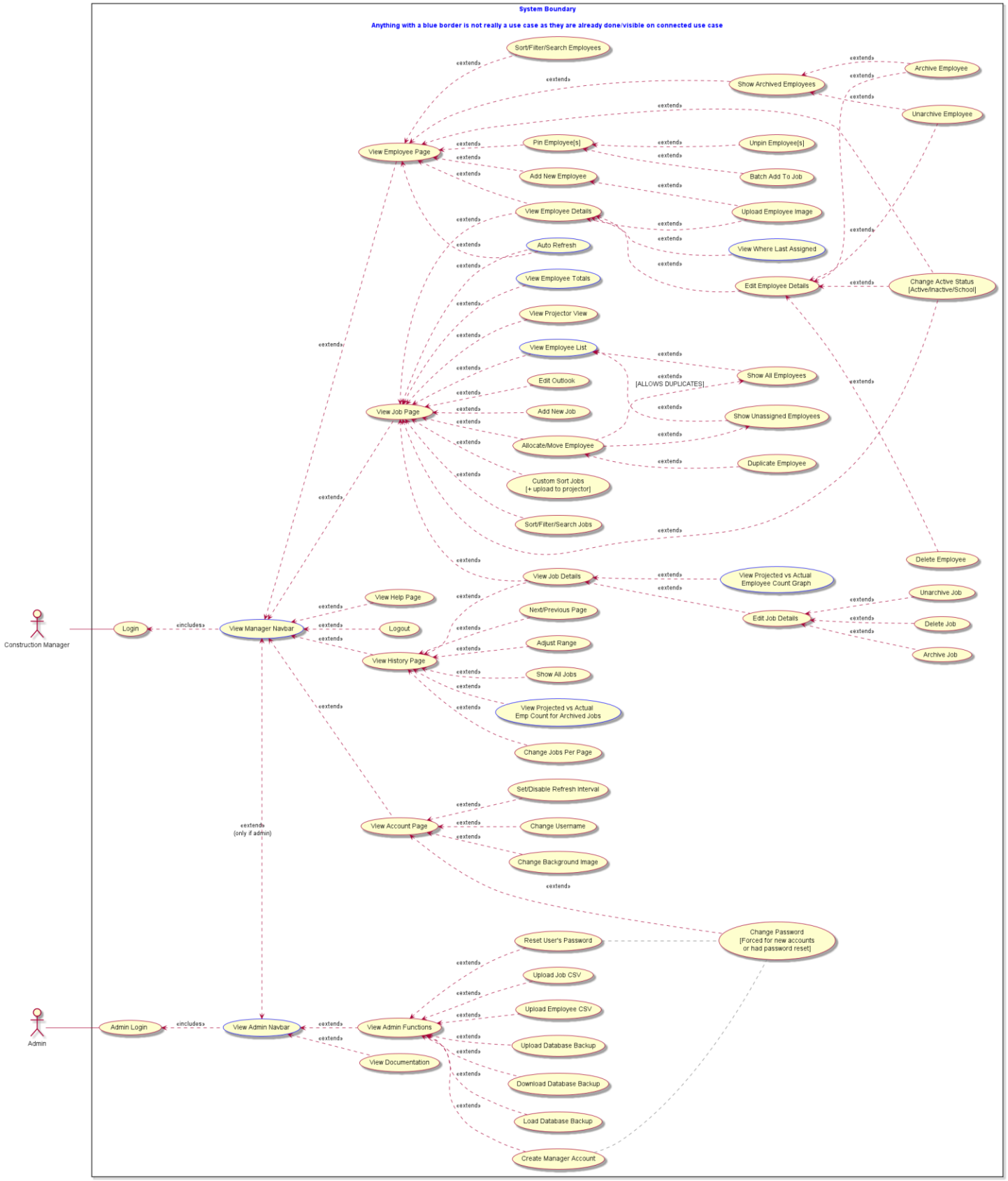
- ***global_data_table.php*** and ***global_data_table.js*** has is a central location that contains an array of all the employee statuses and roles for use in the backend and frontend respectively.

- ***global_generators.php*** has the important *generate_nav_bar()* function that is used on every page for generating the navbar and the currently selected navbar element. It also will show a different navbar for admins. This function also calls the *set_background_css()* file for generating the background on each page.

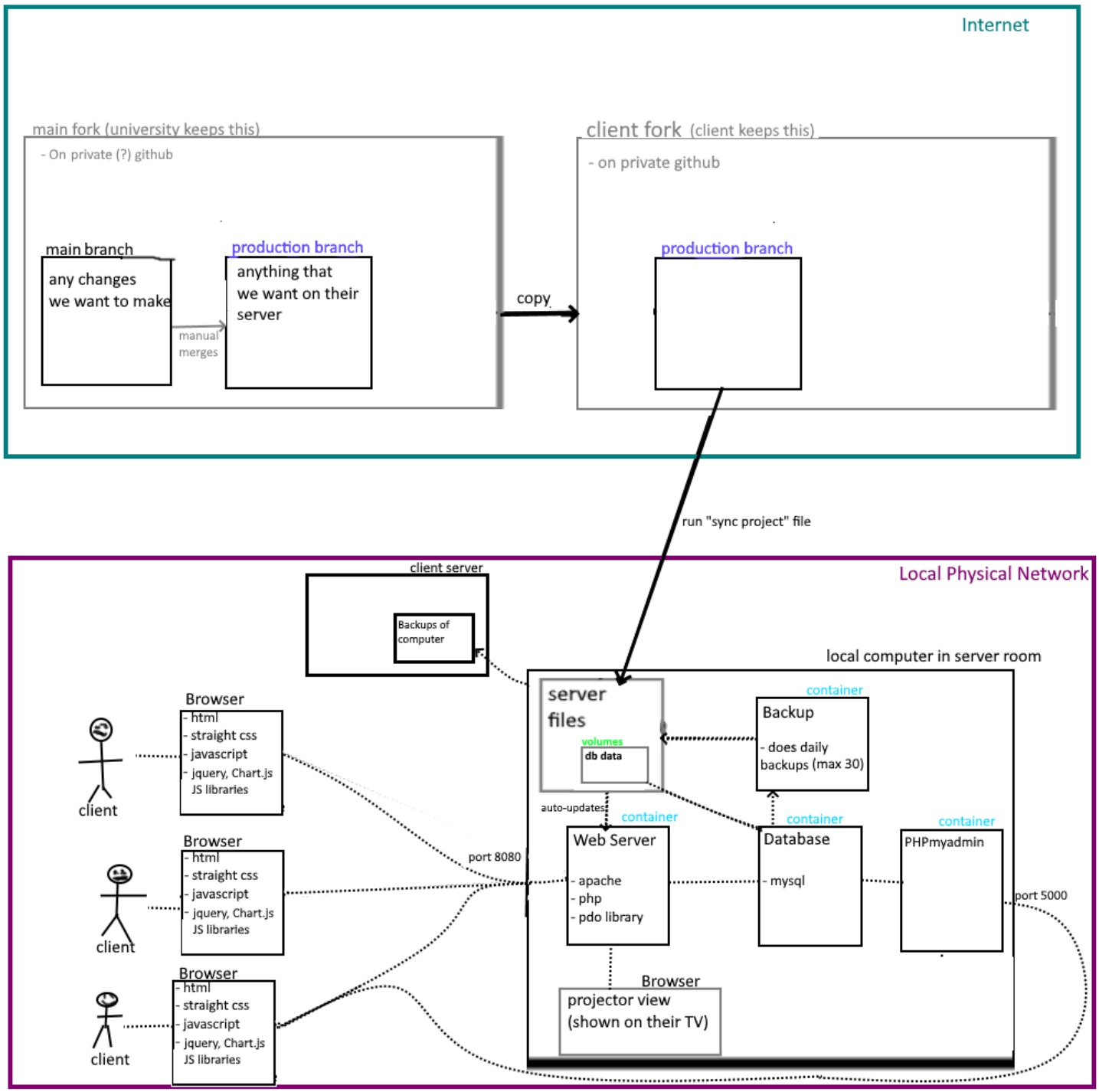
- ***session_security.php*** has some *essential* functions (referenced in nearly every file). It has the *security_check_and_connect()* function whose main job is to first make sure that the user is logged in, and second allows you to pass in what kind of request method (GET or POST) and variables needed to view the page. Despite what the name suggests, this function does not actually connect to the database, but rather importing the *session_security.php* file itself will import the *database-con.php* file, which makes the database connection (sets *$pdo* to the db connection). This file also has the *terminate()* function, which simply closes the database connection, sets the response code to whatever you give it (default 200), and calls php's *die()* with the message you give it. The last function is *check_is_admin()*, which just checks if the admin session variables are set.

- Most of our php files separate will separate using the *terminate()* function and $_POST variables outside of the function. This is because phpunit does not play nice with superglobals and php's die() function (will terminate phpunit while writing tests). If you're looking to make a new backend feature that's testable, *editProjectionNumber.php* is a good place to start.

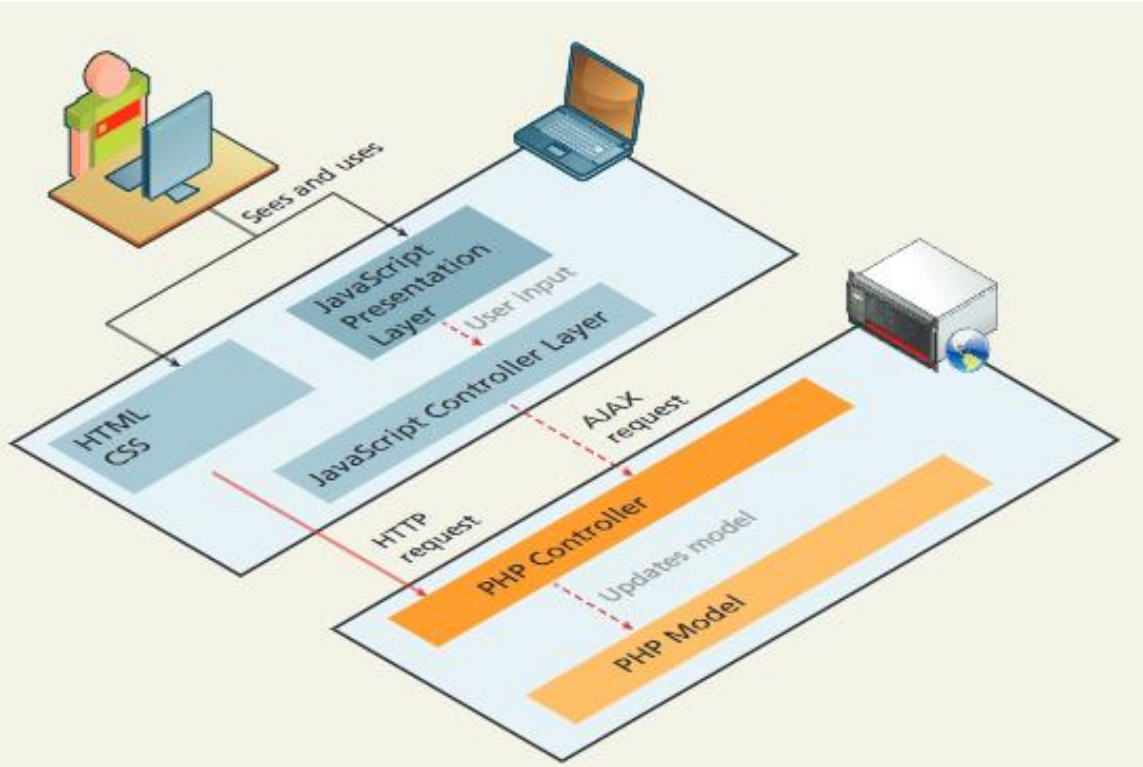
UML Case Diagram

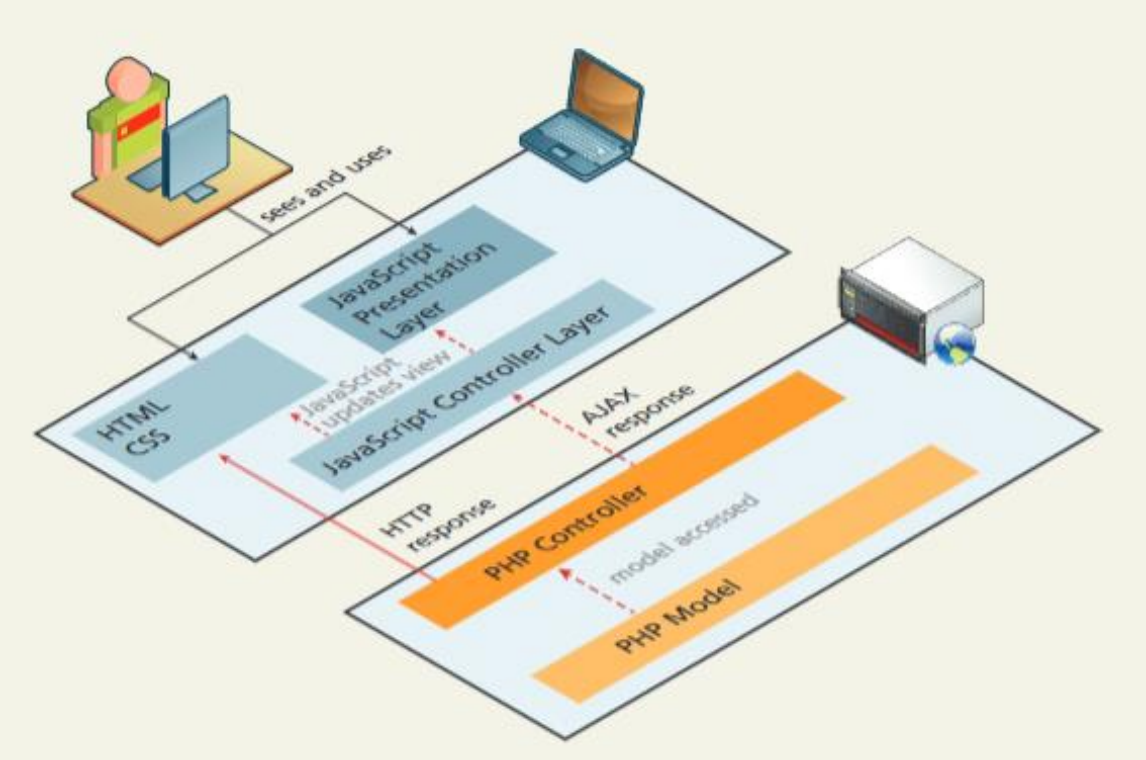


Architecture Diagram (with deployment)



Architecture Diagrams - MVC

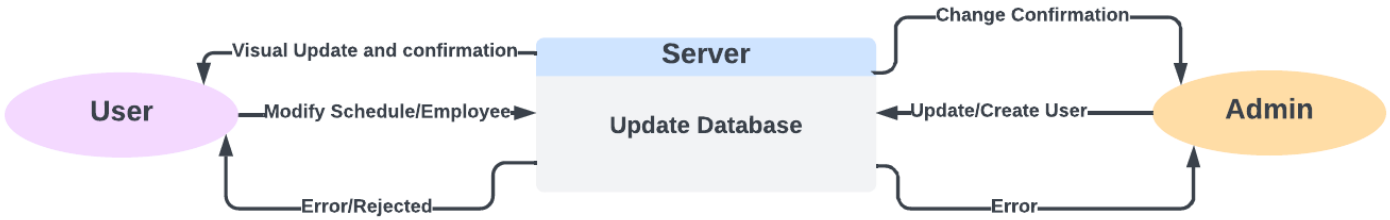




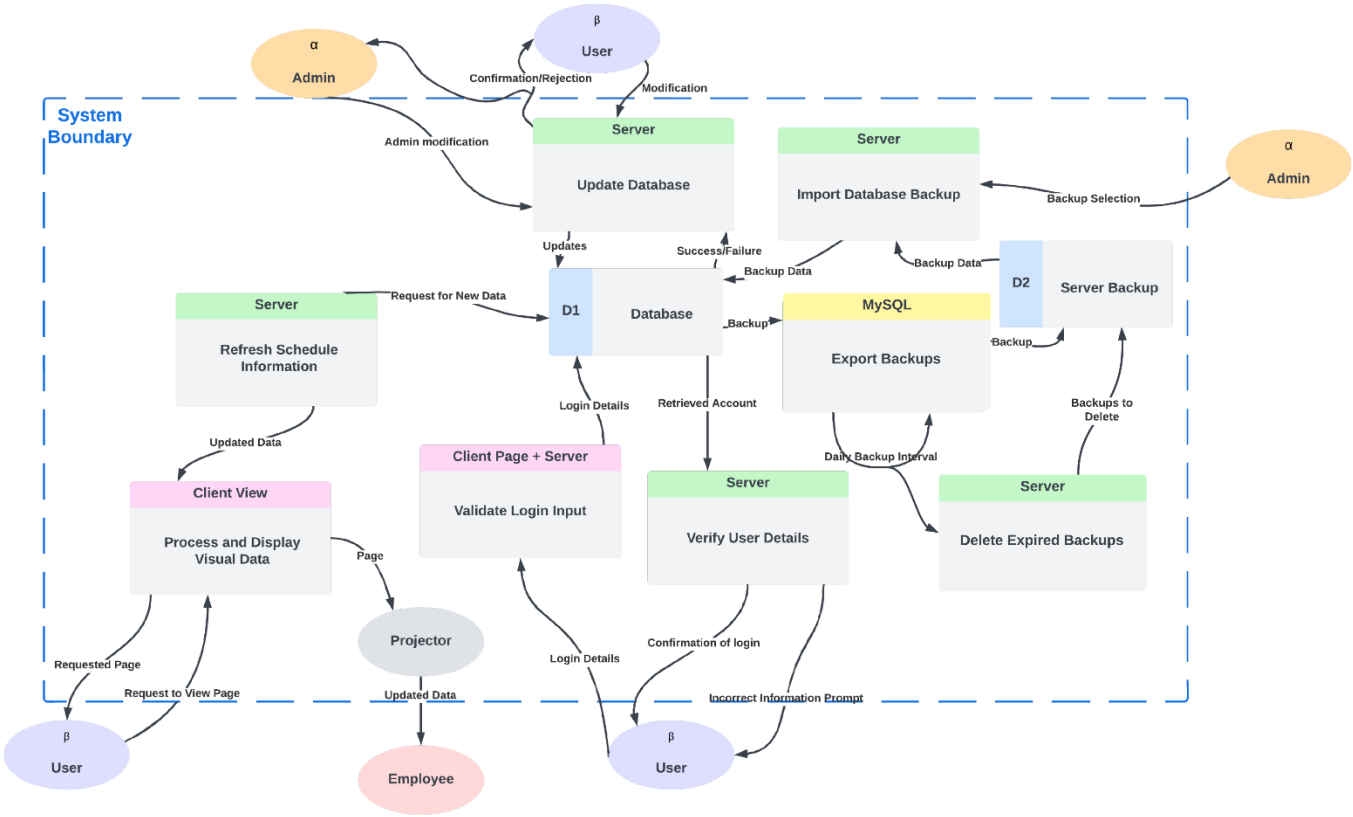
Credits: Figure 17.14/15 from Fundamentals of Web Development 2nd edition, Randy Connolly & Ricardo Hoar, 2018.

Dataflow Diagrams

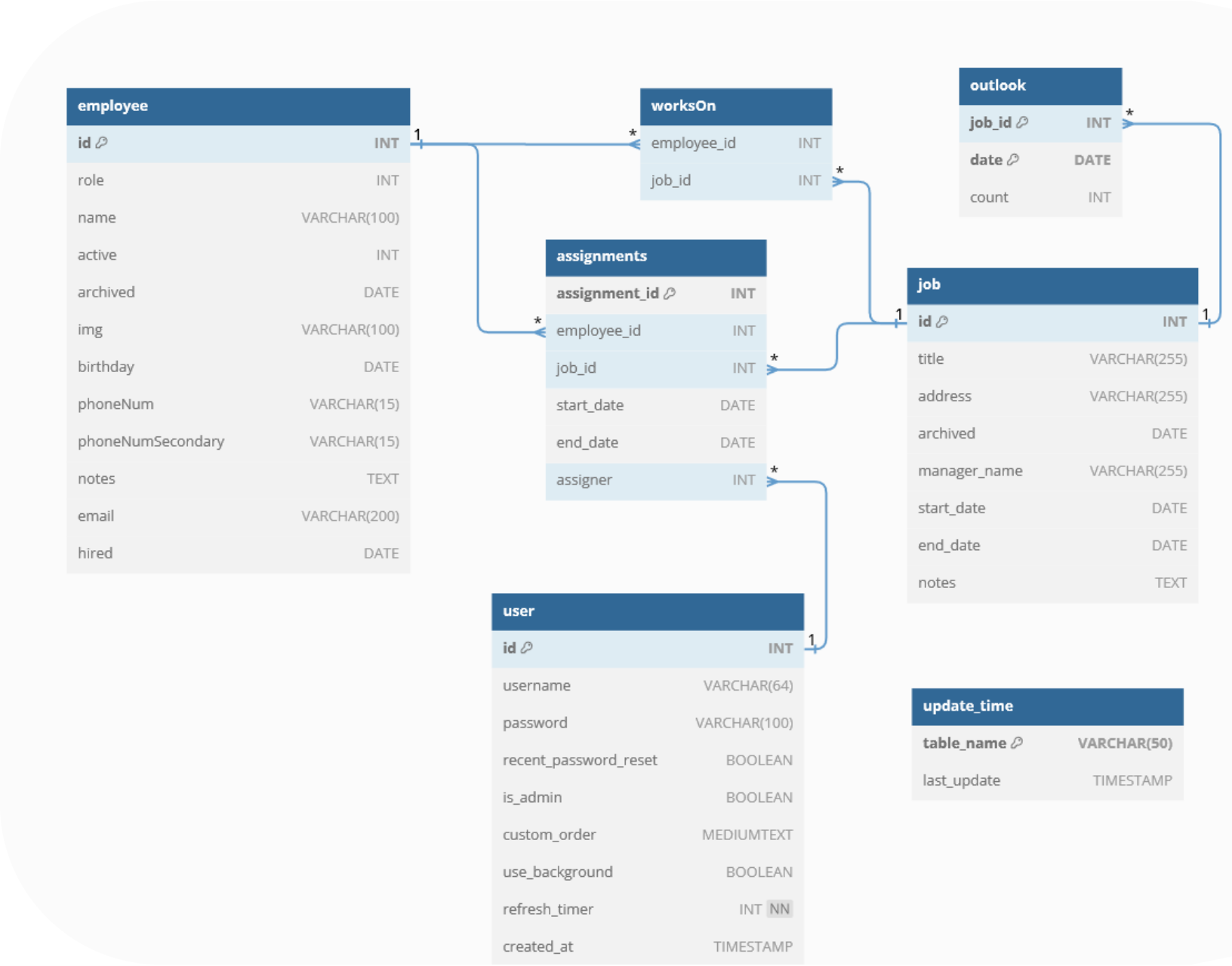
DFD Level 0



DFD Level 1



Database Diagram



Use Cases (+Notable Features)

(highlighted cases were not finished documenting)

Critical Use Cases- Cannot function without

1. View Job Page

Description:

Primary actor: Construction Manager

Description: This use case involves the Construction Manager logging into the system and accessing the "view job page" by clicking on the "job" tab. The purpose is to provide the user with a clear and organized view of job-related information.

Pre-condition: The user (Construction Manager) must be logged in to the system.

Post-condition: The user is presented with the "view job page," displaying job-related information.

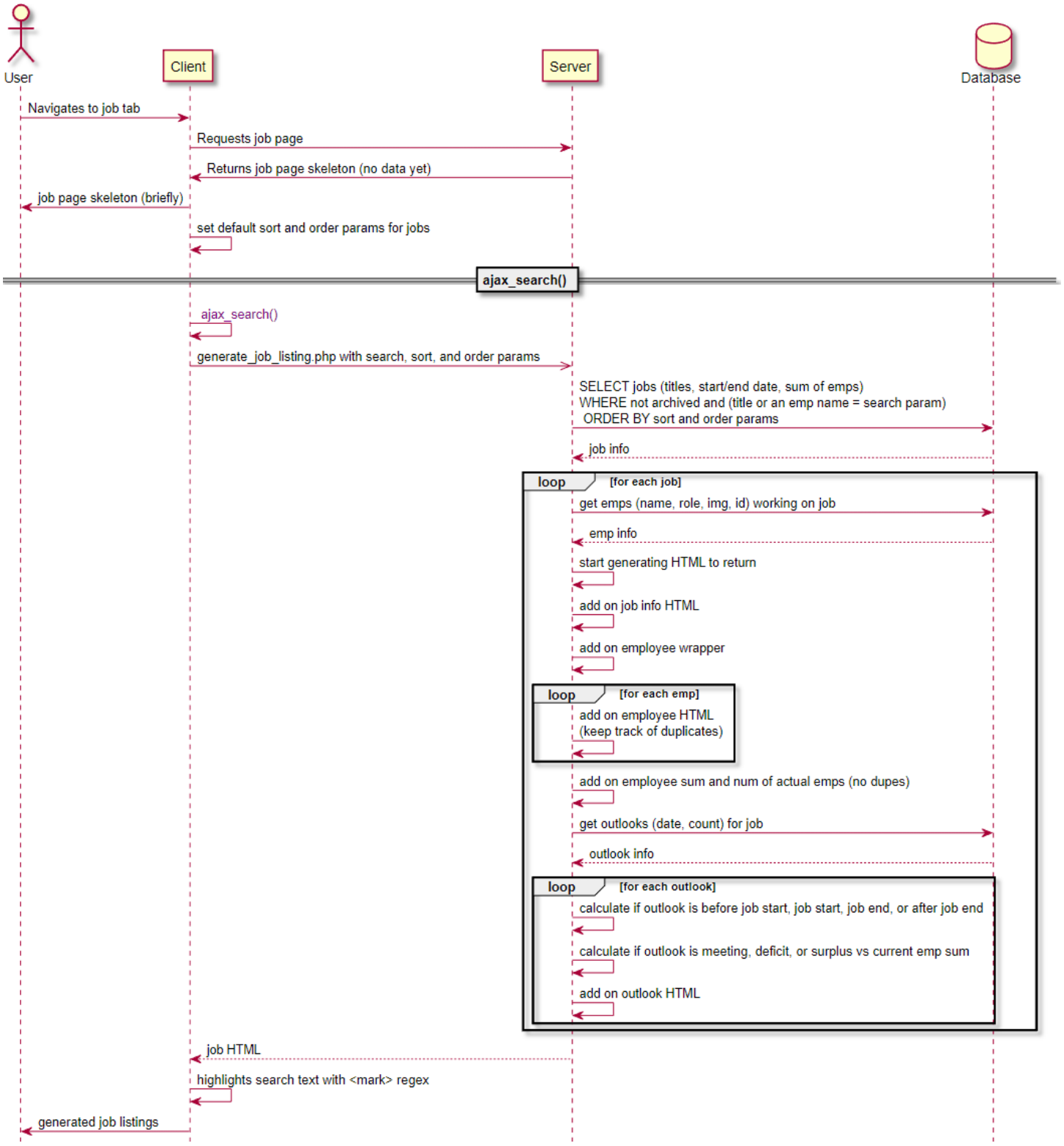
Main scenario:

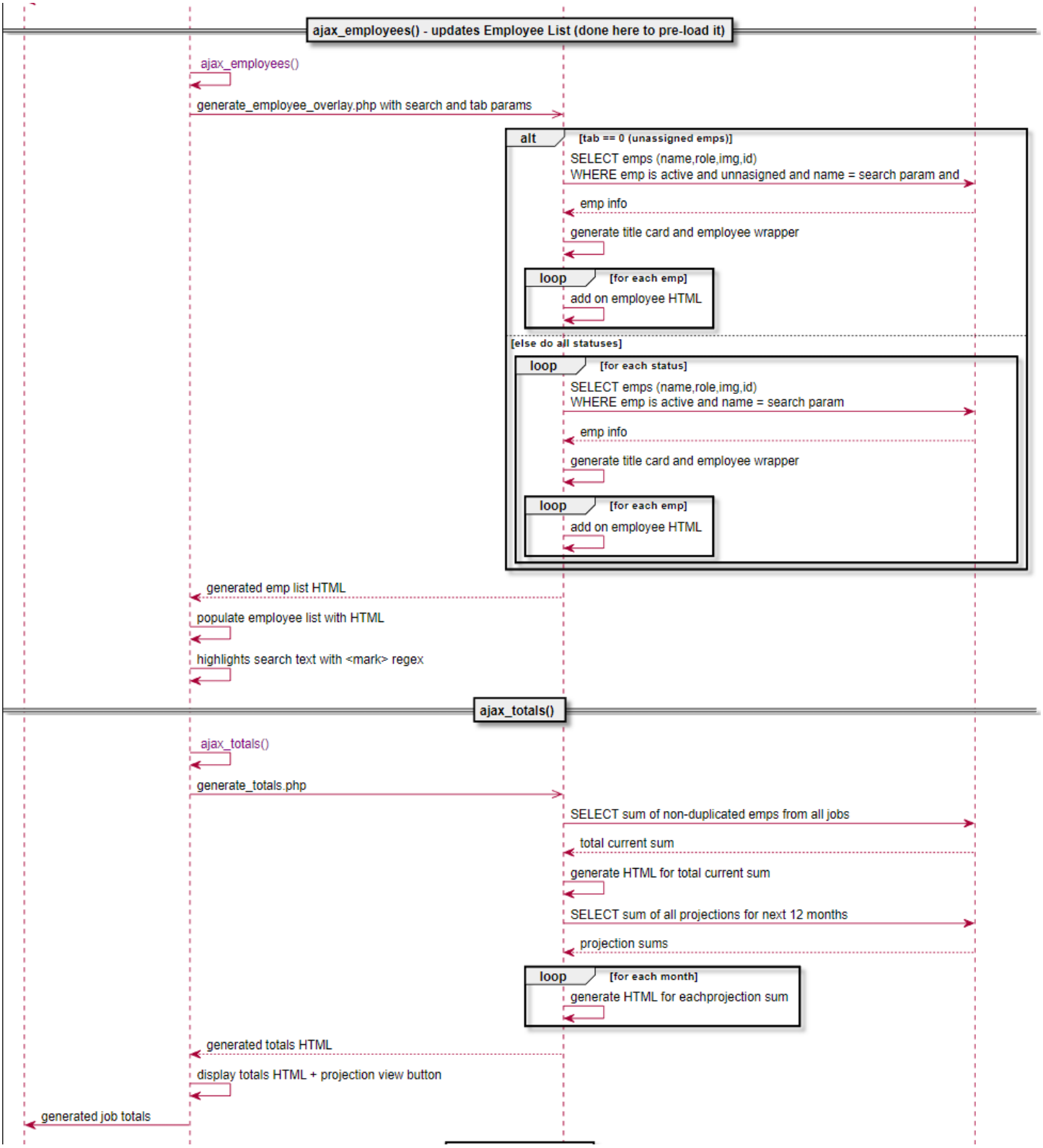
- The user clicks on the "job" tab.
- The system fetches and displays the job-related information on the "view job page."

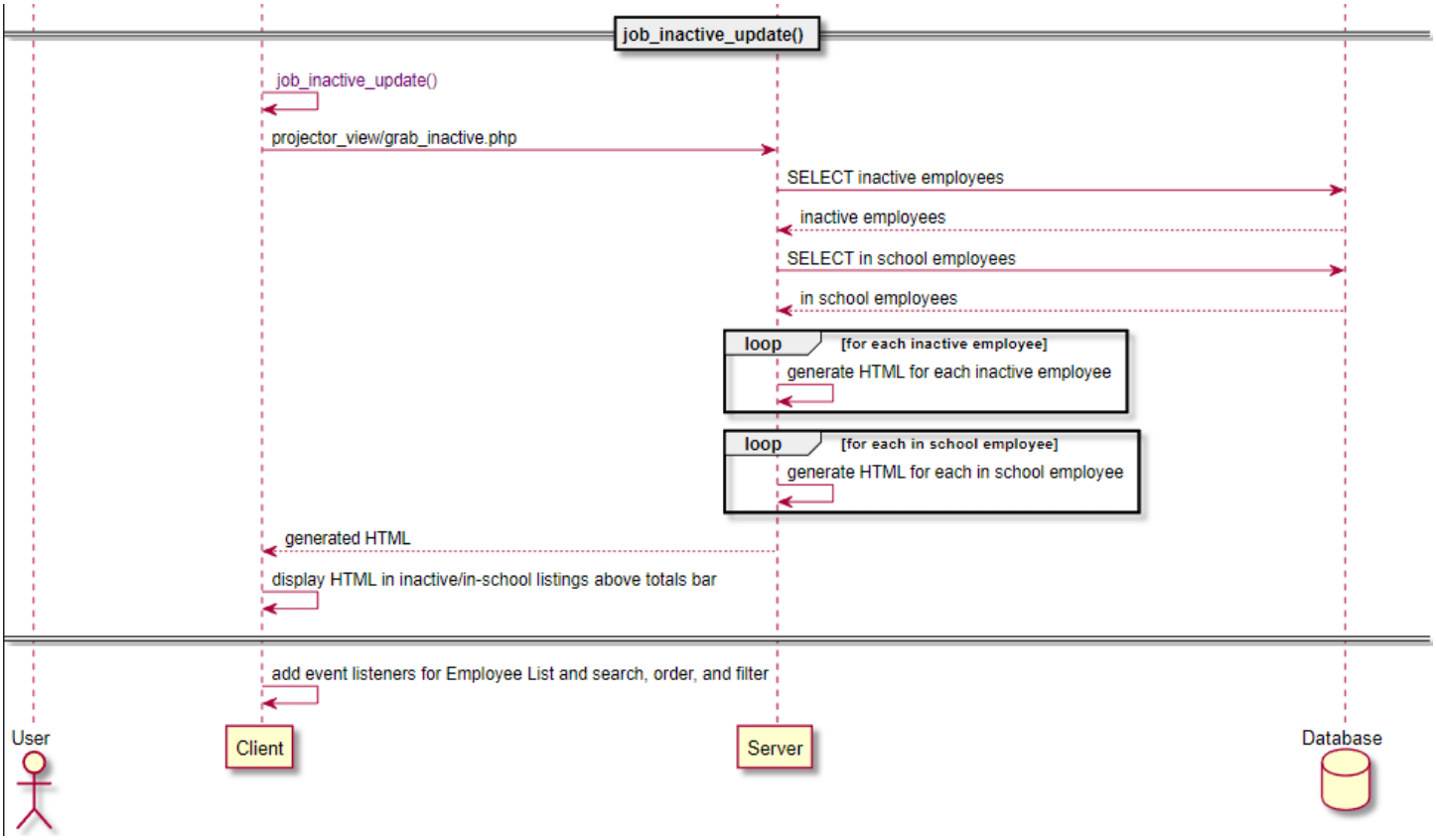
Extensions:

1. No jobs in database
   1.1. Show that there's no jobs found

View Job Page Sequence Diagram




ajax_employees() - updates Employee List (done here to pre-load it)
ajax_employees()
generate_employee_overlay.php with search and tab params
alt
[tab == 0 (unassigned emps)]
SELECT emps (name,role,img,id)
WHERE emp is active and unnasigned and name = search param and
emp info
generate title card and employee wrapper
loop
[for each emp]
add on employee HTML
[else do all statuses]
loop
[for each status]
SELECT emps (name,role,img,id)
WHERE emp is active and name = search param
emp info
generate title card and employee wrapper
loop
[for each emp]
add on employee HTML
generated emp list HTML
populate employee list with HTML
highlights search text with <mark> regex
ajax_totals()
ajax_totals()
generate_totals.php
SELECT sum of non-duplicated emps from all jobs
total current sum
generate HTML for total current sum
SELECT sum of all projections for next 12 months
projection sums
loop
[for each month]
generate HTML for eachprojection sum
generated totals HTML
display totals HTML + projection view button
generated job totals



Upon load of the job page, the page calls *generate_job_listing.php* which queries the database for the list of jobs and employees on them then generates and returns the HTML code for the job page content. Most can be explained through the sequence diagram.

2. Allocate/Move Employee

Primary actor: Construction Manager

Description: Allow the user to easily and quickly move employees from jobs

Pre-condition: The user must be logged in to the system and on the job view page

Post-condition: The employee is moved visually and in the database.

Main scenario:

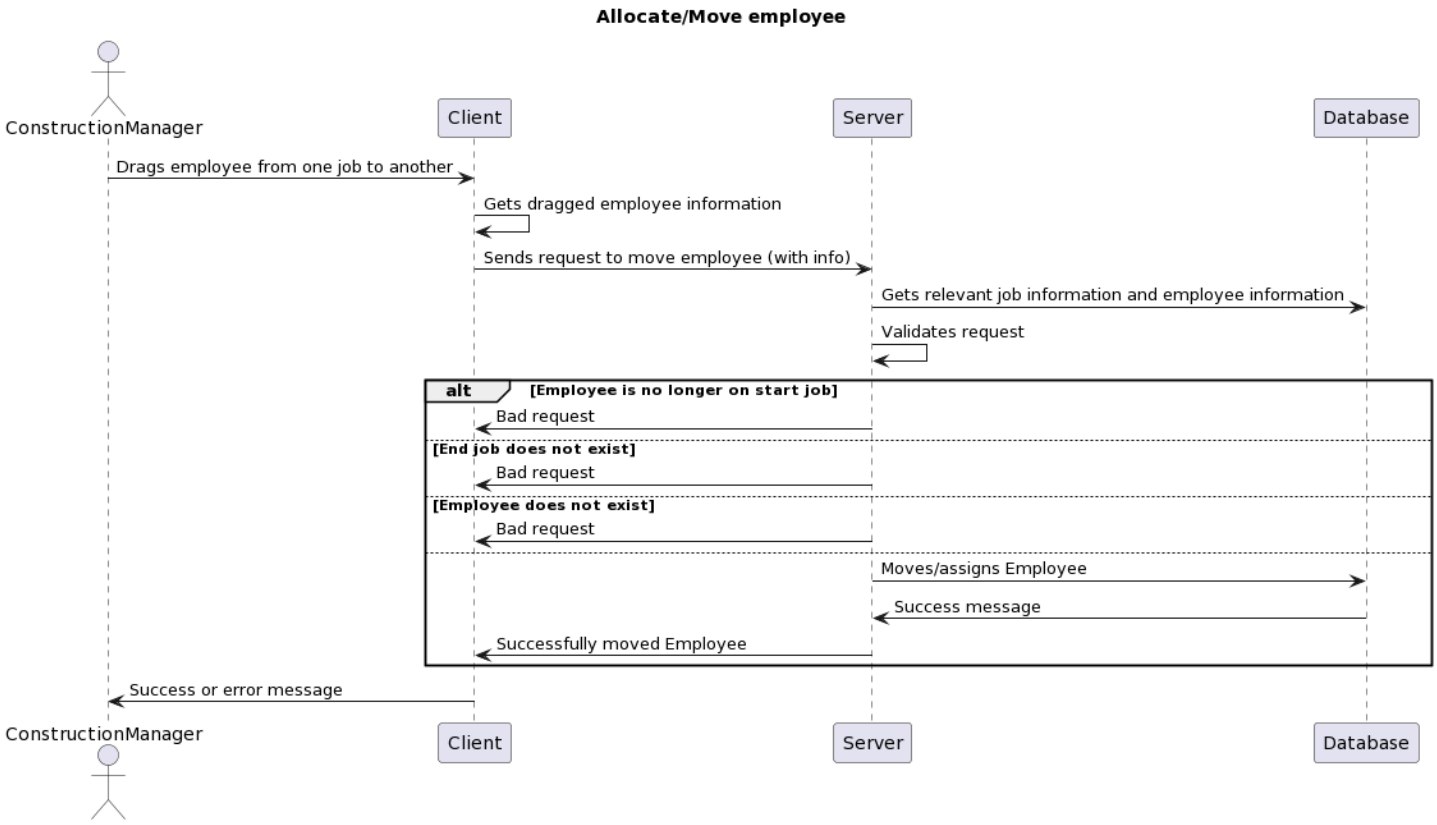
1. User navigates to the 'job' tab
2. User drags employee from job to other job
3. System visually keeps the change.
4. System updates the database with the change

Extensions:

4.1. Database error

4.1.1. Database fails to update with new data due to connection issues or a failed database constraint.

4.1.2. The UI notifies the user that the update has failed and the reason for failure through 'send_message()' function



In *drag-drop.js* in the job view folder, when a drag event is detected, information about the dragged employee, such as their current job assignment, their employee id, and the element that the employee was dragged from, is stored in the drag event. On drop, the information is read and appropriate actions performed by calling *move_employee.php* with an ajax request. *Move_employee.php* performs some checks to make sure that the employee is not being dropped to an invalid target, that the employee and job id's are valid, and that the employee is actually stationed at the job that the drag event recorded. If all of these checks pass, then the employee is moved or assigned to the job. If not, then an error message is returned to the client. If the employee was dropped on the same element as it was dragged from, the action is cancelled. After a successful move or assignment, the script calls *ajax_search()* to refresh the displayed information on the page

3. Add New Job

Primary actor: Construction Manager

Description: Allow the user to add a new job/project to the system. New jobs will require the following fields: Job name, start/end date, project manager (can be displayed as separate on the actual field). New jobs will show on the schedule and will display the job name, the project manager, space for employees, as well as displaying areas which contain the expected number of employees needed for the month (adjusted manually). The schedule will have indicators on where the expected start and end month is.

Pre-condition: The user must be logged in to the system and viewing the job page

Post-condition: The new project is successfully added to the system and the schedule will be updated accordingly.

Main scenario:

1. The user selects the "Add New Job/Project" button from the user interface.

2. The system presents a form to prompts the user to enter the name, duration (start month, end month), and the project manager, and address of the new job/project.

3. The user fills in the required fields.

4. The user submits the form.

5. The system validates the entered information to ensure it meets any defined constraints.

6. The system adds the new job/project to the system.

7. The system stores the job/project details in the database.

8. The system provides feedback to the user indicating the successful addition of the new job/project, and the displayed schedule will be updated accordingly.

Extensions:

5.1. Validation errors

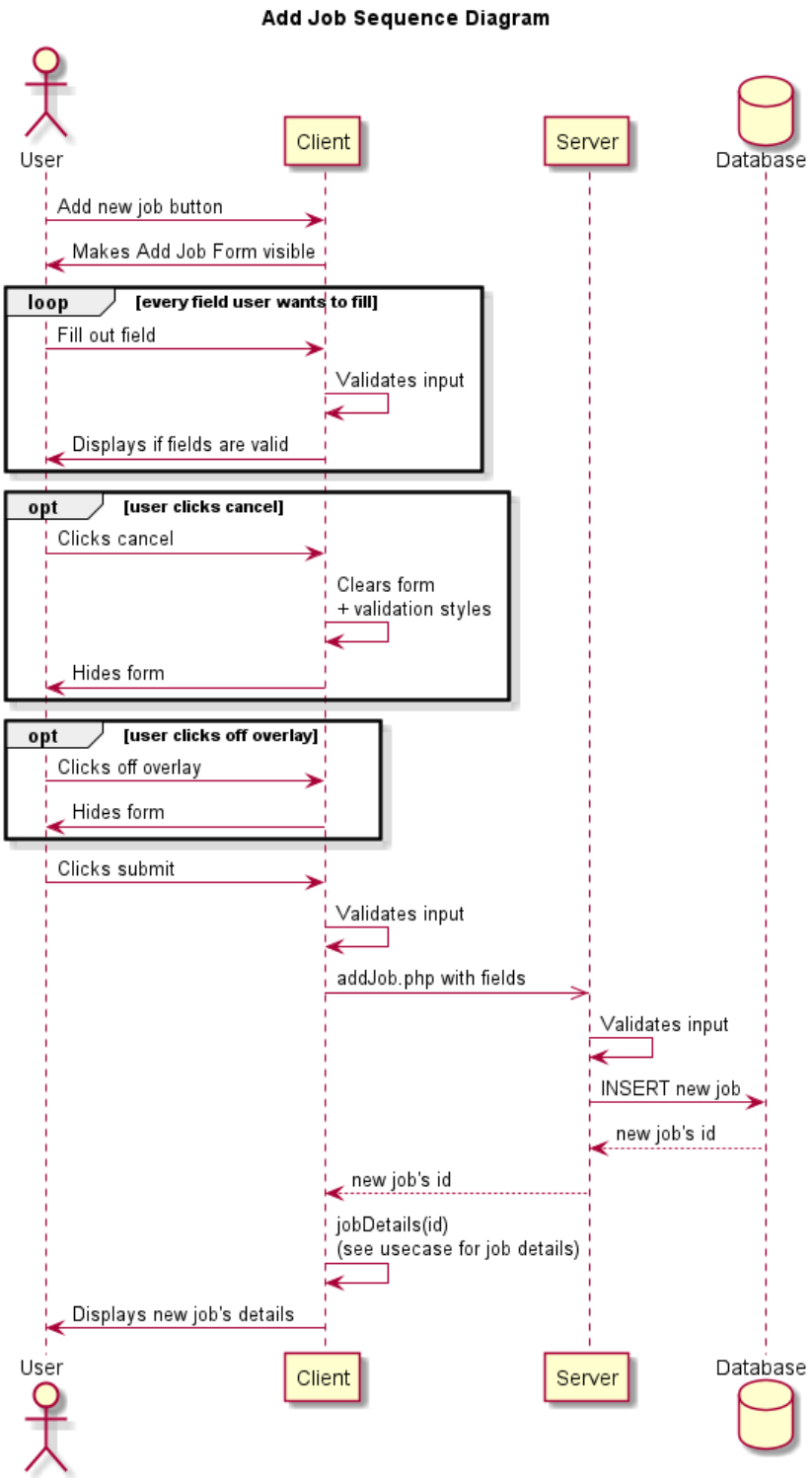
5.1.1. If the system encounters validation errors during the data entry process, such as missing required fields or conflicting data, it notifies the user and provides details about the encountered issues.

5.1.2. The system prompts the user to correct the errors before resubmitting the form.

7.1. Database update errors

7.1.1. If the system encounters errors while adding the new job/project to the database, such as database connectivity issues or conflicts with existing data, it notifies the user and suggests retrying the operation later.

7.1.2. The system ensures data consistency by rolling back any incomplete or failed updates, maintaining the integrity of the job database.



Add job is a little different compared to the other overlays, as is essentially pre-loaded into the page except invisible, where only its visibility is toggled (it was developed before the more streamlined generate/close overlay functions. It was never changed since it easily allows the fields to not be reset when they click off the overlay).

The start date field also gets set to default be this month (can be unset by the user, of course), and that default is set in *job-validation.js*.

Clicking the add job button will toggle its visibility via the *showAddJobModal()* function in *job-view.js*. In *job-validation.js*, 'keyup' event listener is added to every field in the add job overlay which will run their corresponding validation functions. These functions (*validateJobTitle()*, *validateJobManager()*, etc.) will add classes to show that a field is

correct/incorrect, and set the given 'error' element some text describing the error. All of the input fields will also be ran through these validation functions a second time when the user clicks the 'submit' button.

If validation was successful, the form data will be sent to *addJob.php*, which performs some more validation and returns the new job's id. This id then gets retrieved by the client, which will show the *jobDetails()* for that new job id (see job details use case).

The only real error checking cases is if there is no job title, if start date is after end date, and if any of the fields is too long.

4. Add New Employee

Primary actor: Construction Manager

Description: Allow the user to add a new employee to the system.

Pre-condition: The user must be logged in to the system.

Post-condition: The new employee is successfully added to the system.

Main scenario:

1. The user selects the "Add New Employee " button from the user interface.
2. The system presents a form to prompts the user to enter the details of the new employee.
3. The user fills in the required information for the new employee, such as name, address, journeyman level, and other relevant information.
4. The user submits the form.
5. The system validates the entered information to ensure it meets any defined constraints.
6. The system adds the new employee to the system.
7. The system stores the employee details in the database.
8. The system provides feedback to the user indicating the successful addition of the new employee.

Extensions:

5.1. Validation errors

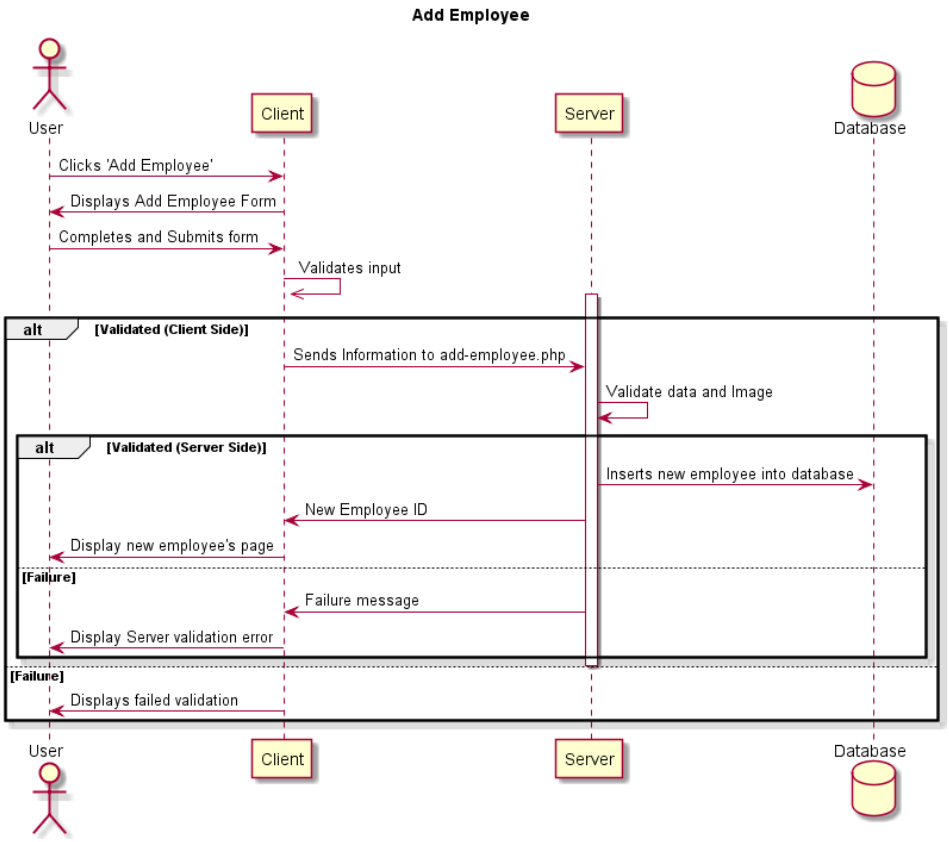
5.1.1. If the system encounters validation errors during the data entry process, such as missing required fields or conflicting data, it notifies the user and provides details about the encountered issues.

5.1.2. The system prompts the user to correct the errors before resubmitting the form.

6.1. Database update errors

6.1.1. If the system encounters errors while adding the new job/project to the database, such as database connectivity issues or conflicts with existing data, it notifies the user and suggests retrying the operation later.

6.1.2. The system ensures data consistency by rolling back any incomplete or failed updates, maintaining the integrity of the job database.



When a user clicks on the Add Employee button *Employee Page/add-employee.js* calls the *generate_overlay()* function inside *global-functions.js* and then populates it with data from *Employee Page/add-employee-overlay.php.*

Once the user finishes filling out the form and clicks submit *Employee Page/add-employee.js* checks to see that required fields are filled, that phone numbers are in the proper format and that no fields are over the database field limits. If this succeeds the data is passed to *Employee Page/add-employee.php* which checks these a second time. If an image is uploaded this file also calls *employee info overlay/fileUpload.php: image-validate()* which checks that the image is of the correct type and that it is not too large. If the validation passes, then the image is compressed and uploaded using *employee info overlay/fileUpload.php: save_user_image()*.

Once this is completed Employee Page/add-employee.php inserts the new employee into the database and returns its new ID. The client then opens the new employee's detail overlay.

5. Edit Outlook (Projection)

Primary actor: Construction Manager

Description: Allows the user to edit the number of employees expected for a specific job for a particular month.

Pre-condition: User must be logged in to the system (#18) and on the job page (#1).

Post-condition: The number of employees assigned to the job for the specified month is successfully updated in the system.

Main scenario:

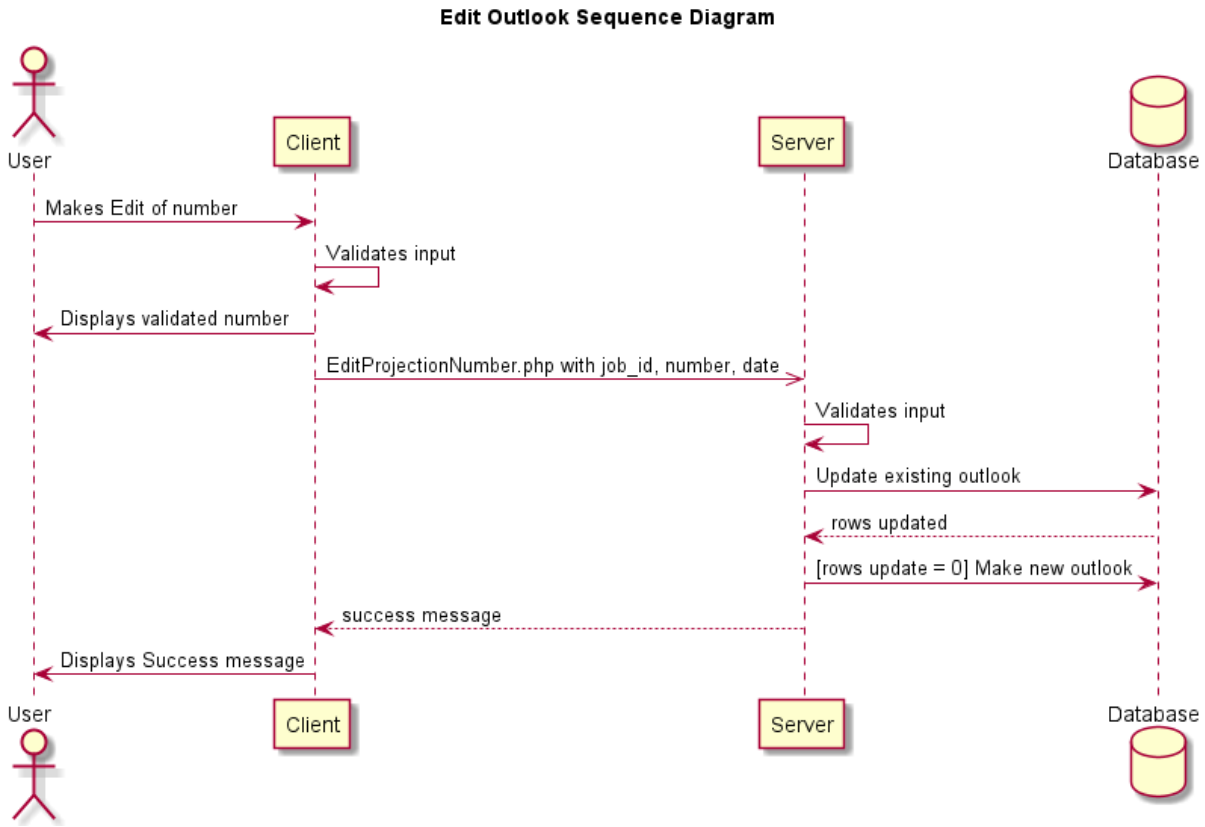
1. User finds the job which they wish to edit
2. User indicates (click arrows, click and hold arrows, type number) to change the projection number for a month.
3. The system validates inputs and updates the number of employees allocated for the job month.
4. System Gives confirmation of update.

Extensions:

2.1. Invalid input format

2.1.1. The user uses a non-integer type as the input. The system will revert the projection to its previous number

Sequence Diagram



All of the projection numbers for a job get printed as number fields in *generate_job_listing.php* which is printed to the page with *ajax_search()*. Once the input boxes are on screen (i.e. right after *ajax_search()* populates the page), the *makeAllProjectionsEditable()* function (in *job-view.js*) is called to add an onchange event listener to all of the projection boxes that will validate the user input and send it to the server with the job_id, the month, and the new projection number.

The way the client-side validation works is before the count is validated, it will save the projection's old value (on focus), then the client validate to make sure that if it is a number and not empty, and if not, the old saved value will be loaded.

Once the server validates the input, it will attempt to update any existing outlooks in the database (projections that have never been edited before are not saved as a way to save space). If no outlooks were updated, it will create a new one.

Once the server responds with the success message, the client will display it and will run *ajax_totals()* to re-calculate the totals bar.

Error checking on client (will revert to old value):

- Outlook number must not be empty, null, or NaN

Error checking for server (will send error message):

- Outlook number must be a number
- Outlook number must not be too big (less than ~2 billion)
- Date format must be correct (YYYY-MM-01)

6. View Employee List (Notable Feature) (**NEW**)

Description:

Primary actor: Construction Manager/User

Description: This use case involves the Construction Manager or User interacting with the system by clicking on the "Employee List" button. The process starts with the user's action triggering a jQuery AJAX call from the client to the server, where the request is sent to the "*generate_employee_overlay.php*" script. This script is responsible for retrieving employee data from the database, generating HTML content based on the data, and returning the HTML back to the client. Upon receiving the HTML, the client dynamically inserts it into the designated element on the interface. As a result, the user is presented with the list of employees in the form of an interactive and visually organized display.

Pre-condition: The user (Construction Manager/User) must be logged in to the system.

Post-condition: The user is presented with an employee list displayed on the client interface.

Main scenario:

The user clicks on the "Employee List" option.

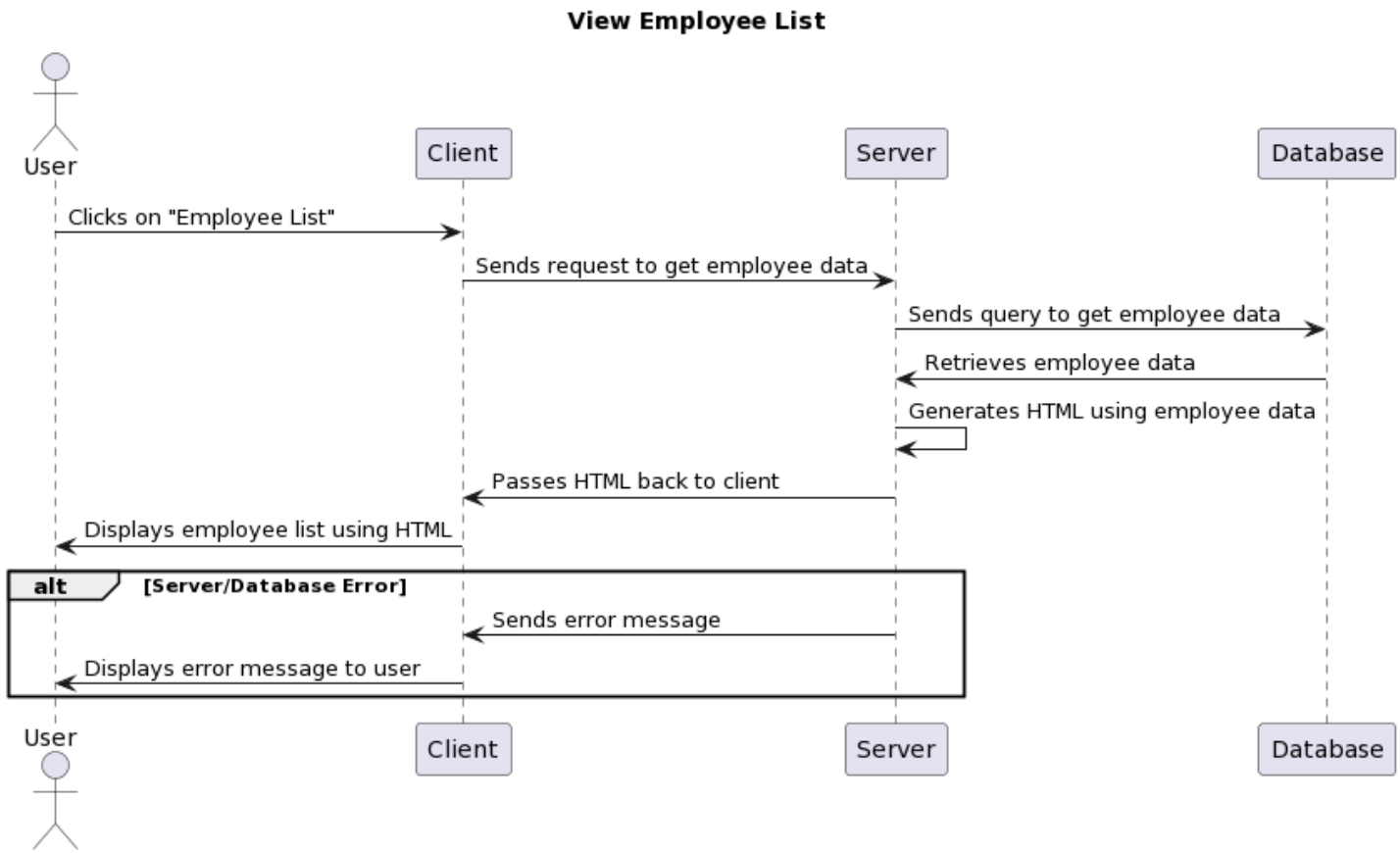
The system displays the list of employees

Extensions:

6.1. Server/Database Error

6.1.1. If there are errors in the server's processing or issues with the database retrieval, the server sends an error message to the client.

6.1.2. The client displays the error message to the user, indicating the nature of the problem.



7. Show All/Unassigned Employees- Job page- via employee list (**NEW**)

Primary actor: Construction Manager

Description: Allow the user to view all unassigned employees at a glance, as well as view all employees if necessary.

Pre-condition: The user must be logged in to the system (use case 5) and viewing the editable schedule (use case 7).

Post-condition: The list of employees is visible and interactable on the editable job view page.

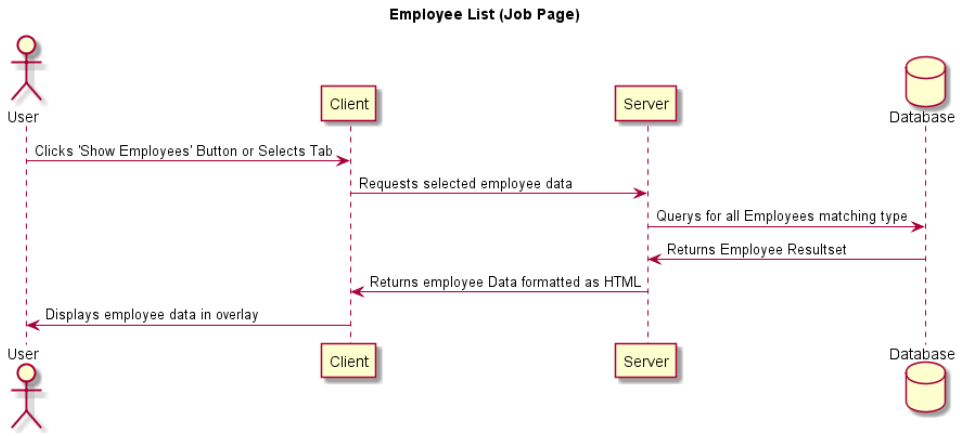
Main scenario:

1. The user clicks on the 'Employee List' button.
2. The system opens the list and populates the results
3. The user may then Move/Allocate an employee (Use Case 2) or View Employee Details (Use Case 8)

Extensions:

2.1. Database fetch errors

2.1.1. If the system encounters errors while fetching employee details in the database, such as database connectivity issues or conflicts with existing data, it notifies the user.



When a user clicks on 'show employees' j*ob-view/job-view.js: ajax_employees()* requests data from *job-view/generate_employee_overlay.php.* This file queries the database for all employees that meet the criteria and returns it as formatted HTML. *ajax_employees()* then displays the data as an overlay.

8. Change Active Status [Active/Inactive/School] (**NEW**)

Primary actor: Construction Manager

Description: Change the active status of an employee to reflect their current status i.e. Inactive if they are on medical leave or vacation, School if they are in school or active if they are working/available to work.

Pre-condition: The user must be logged in to the system (use case 5) and either on the Employee Page (Use Case 25), on the Job page (Use Case 1) or editing employee Details (Use Case 9).

Post-condition: The Employee's status is updated in the database and dependant data is also updated.

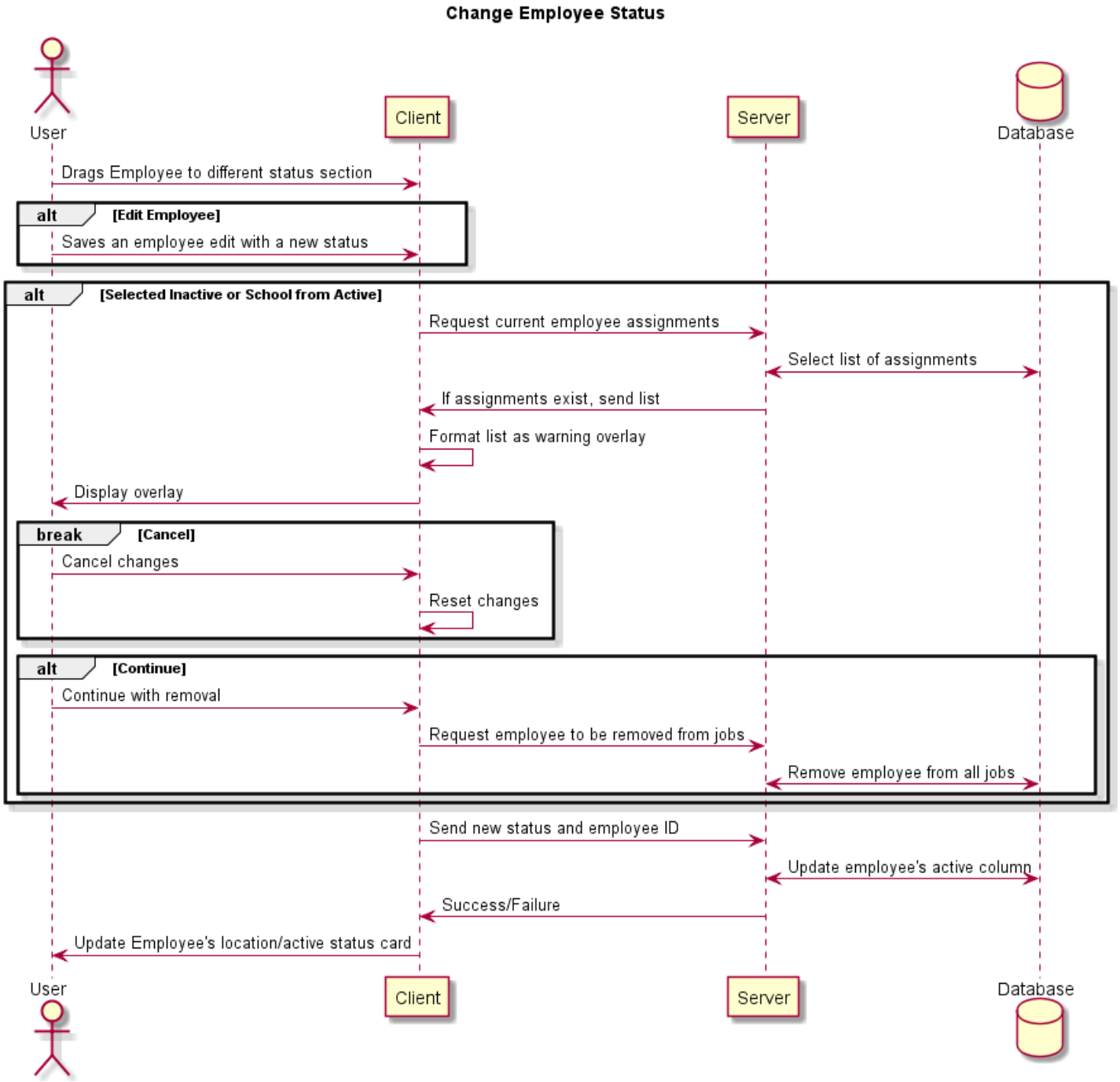
Main scenario:

1. The user changes selects a change to the employees status (Dragged into area on job or employee view/Changed in the edit employee view)
2. The system checks what else this may effect i.e. employee cant be inactive or in school and assigned to a job
3. If the employee has conflicting dependant variables the system will notify the user that these will be overwritten
4. The User may confirm to overwrite these dependencies or can cancel the status change

Extensions:

2.1. Database fetch errors

2.1.1. If the system encounters errors while fetching the employee's details in the database, such as database connectivity issues, it notifies the user and suggests retrying later.



Users may edit employee status in one of two ways; dropping their card inside a different status box on *the job-view.php* or *employees.php* pages OR by changing the status while editing an employee. The result and user experience for these are nearly identical but the functionality is slightly different.

For the drag and drop function on the employee page *Employee Page/drag-drop.js-drop_employee()* is called, however, on the job page *job-view/drag-drop.js-active-drop-employee()* is called. These functions are nearly identical, the job view one can just handle the slightly different variable structure present on the page.

In the case of editing an employee, if the drop-down menu is changed *employee info overlay/employee-info.js-saveEdit()* awaits the promise from *employee_assignment_handler()* which functions identically as the other functions but instead of continuing the function, it resolves the promise with either a true for a confirm delete or false for cancel.

These functions all check where the employee was initially and where they have been moved to. If an employee has been moved from active to inactive/in school *Employee Page/check_emp_job_assignments.php* will be called with

an ajax function. This page checks to see if an employee is assigned to any jobs, if they are it will return an array of job listings. The javascript function will then create a prompt for the user that asks them to confirm that they wish to remove the employee from the listed jobs.

If the user confirms this prompt *Employee Page/mass_remove.php* will then be called using an AJAX function. This function removes the employee from each job. After this the javascript will then call *employee info overlay/employee-info.js-change_active()* which in turn calls *Employee Page/change-active.php* which will change their active status in the database.

In the case of a change from inactive/in school to active the *change_active()* function will be called immediately.

9. View Employee Details (**NEW**)

Primary actor: Construction Manager

Description: Allow the user to view employee details and recent assignments.

Pre-condition: The user must be logged in to the system (use case 5) and viewing the editable schedule (use case 7) or employee page (Use case 25).

Post-condition: The Employee details are populated and visible.

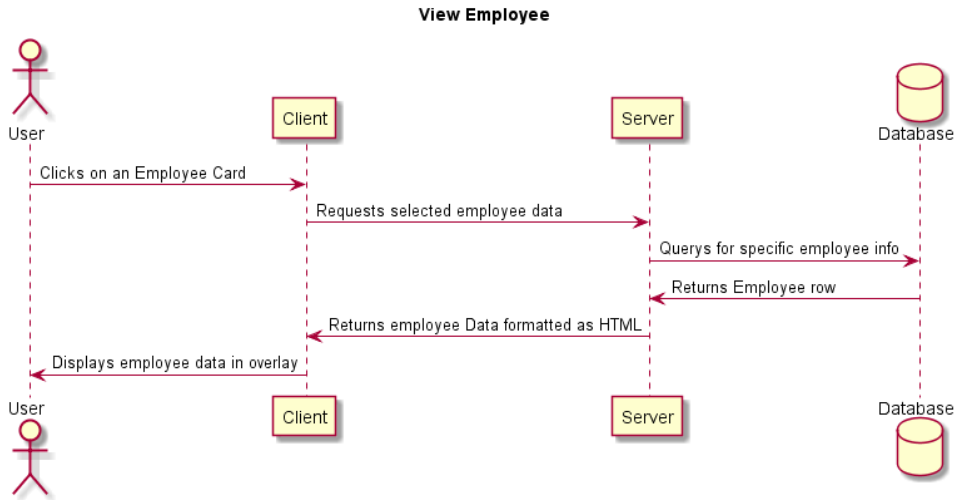
Main scenario:

5. The user clicks on the employee they wish to view
6. The system grabs all relevant information from the database
7. The overlay is populated with this information
8. The User may edit the saved employee details or close the overlay.

Extensions:

2.1. Database fetch errors

2.1.1. If the system encounters errors while fetching the employee's details in the database, such as database connectivity issues, it notifies the user and suggests retrying later.



When an employee card is clicked *employee info overlay/employee-info.js: employeeDetails()* calls *employee info overlay/emp-info-overlay.php* which selects the employee information from the employee table in the database.

It then returns this data formatted as HTML to the client. The client then uses *global-functions.js: generate_overlay()* to create the container and fills it with the supplied data

10. Edit Employee Details

Primary actor: Construction Manager

Description: Allow the user to edit employee details in the system. The user will be able to change the employee's name, their experience level (foreman, superintendent, journeyman, 4^{th}, 3^{rd}, 2^{nd}, 1^{st}, etc.) and whether or not they are active

Pre-condition: The user must be logged in to the system (use case 5) and viewing the editable schedule (use case 7).

Post-condition: The employee details are successfully updated in the system. If the employee is changed from active to inactive (or vise-versa) or if the employee's experience level is changed, the system displays as such.

Main scenario:

4. The user selects the employee whose details they want to edit.
5. The system displays the current details of the selected employee, such as name, address, whether or not they are active, experience level, red seal status, and other relevant information.
6. The user enables editing.
7. The user makes the necessary edits to the employee's details.
8. The user saves the changes.
9. The system validates the updated information to ensure it meets any defined constraints.
10. The system updates the employee's details in the system.
11. The system provides feedback to the user indicating the successful update of the employee's details.

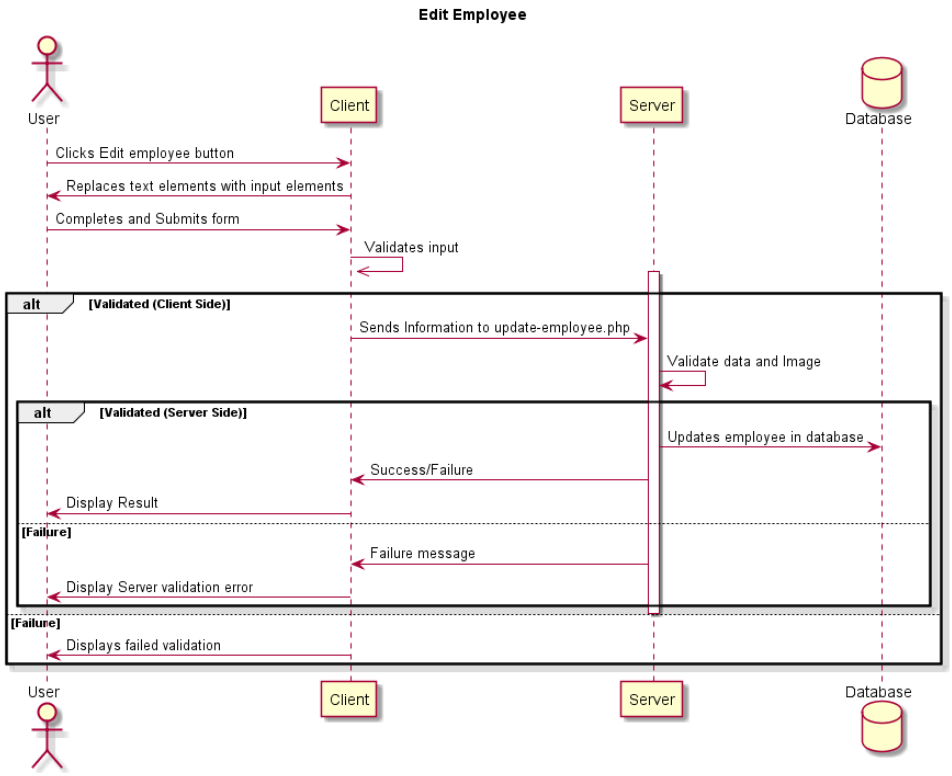
Extensions:

5.1. Validation errors

5.1.1. If the system encounters validation errors during the update process, such as invalid information format or conflicting data, it notifies the user and provides details about the encountered issues.

6.1. Database update errors

6.1.1. If the system encounters errors while updating the employee's details in the database, such as database connectivity issues or conflicts with existing data, it notifies the user and suggests retrying the update later.



When the edit button is clicked while viewing an employee *employee info overlay/employee-info.js: enableEdit()* replaces all text fields with input fields and adds a save button to the page. When the user clicks submit *employee info overlay/employee-info.js: saveEdit()* then validates that all constraints are proper (required fields, length constraints, phone number format) and then sends the data to *employee info overlay/update_employee.php*. This file then validates the inputs once again and if an image is uploaded this file also calls *employee info overlay/fileUpload.php: image-validate()* which checks that the image is of the correct type and that it is not too large. If the validation passes, then the image is compressed and uploaded using *employee info overlay/fileUpload.php: save_user_image()*.

If all the checks pass *update_employee.php* then updates the employees' row in the database and sends the result back to *employee-info.js*. The overlay is then rebuilt with the new data.

11. View Job Details (**NEW**)

Primary actor: Construction Manager

Description: Allow the user to view all relevant information about a given job.

Pre-condition: The user must be logged in to the system (use case 5) and viewing the editable schedule (use case 7).

Post-condition: The job information is shown on top of the editable schedule

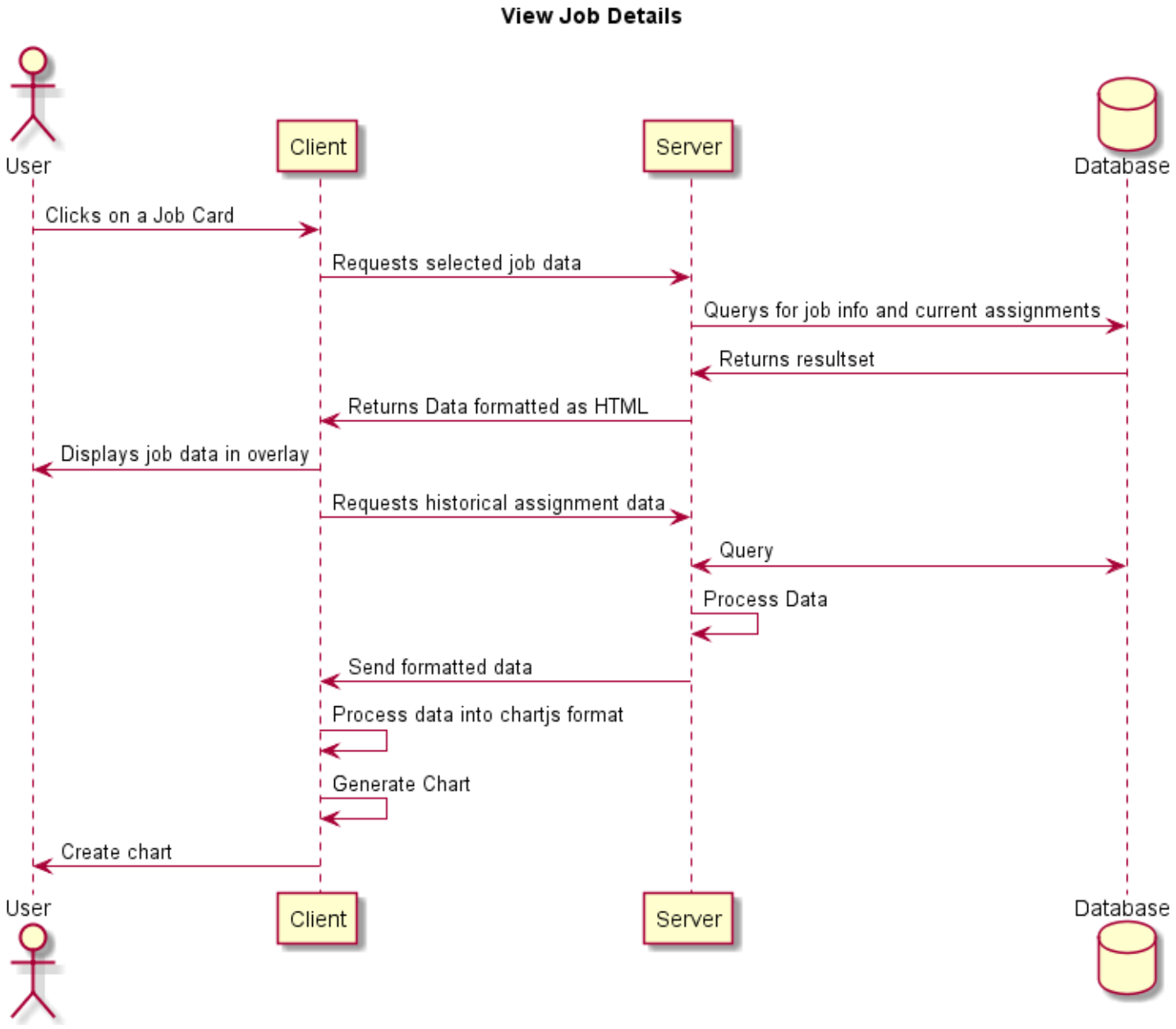
Main scenario:

1. The user clicks on the job details section of the listing
2. The system opens the overlay and populates it with job details as well as historical information and current assigned employees.
3. The user may then Edit Job Details (Use case 11)
4. The user may click on the button again or click elsewhere on the page and close the overlay.

Extensions:

2.1. Database update errors

2.1.1. If the system encounters errors while fetching details in the database, such as database connectivity issues or conflicts with existing data, it notifies the user.



When a user clicks on a job card *job-details/job-info.js-jobDetails()* makes an ajax call to *job-details/job-detail-overlay.php* which query's the database for all information relating to the job and formats it as HTML. This is then put displayed in an overlay wrapper.

jobDetails() then calls *generate_graph()* which makes an ajax call to *job-details/get-job-graph-data.php* and querys for historical data for a specific job. *Generate_graph()* then calls *outlook_graph()* which formats the data in the proper format and creates a chartJS graph which is then placed in the overlay.

12. Edit Job Details

Primary actor: Construction Manager

Description: Allow the user to edit job/project details in the system.

Pre-condition: The user must be logged in to the system.

Post-condition: The job/project details are successfully updated in the system.

Main scenario:

1. The user selects the job/project whose details they want to edit.
2. The system displays the current details of the selected job/project, such as name, start and end date, and other relevant information.
3. The user enables editing.
4. The user makes the necessary edits to the job's/project's details.
5. The user saves the changes.
6. The system validates the updated information to ensure it meets any defined constraints.
7. The system updates the job's/project's details in the system.
8. The system provides feedback to the user indicating the successful update of the job's/project's details.

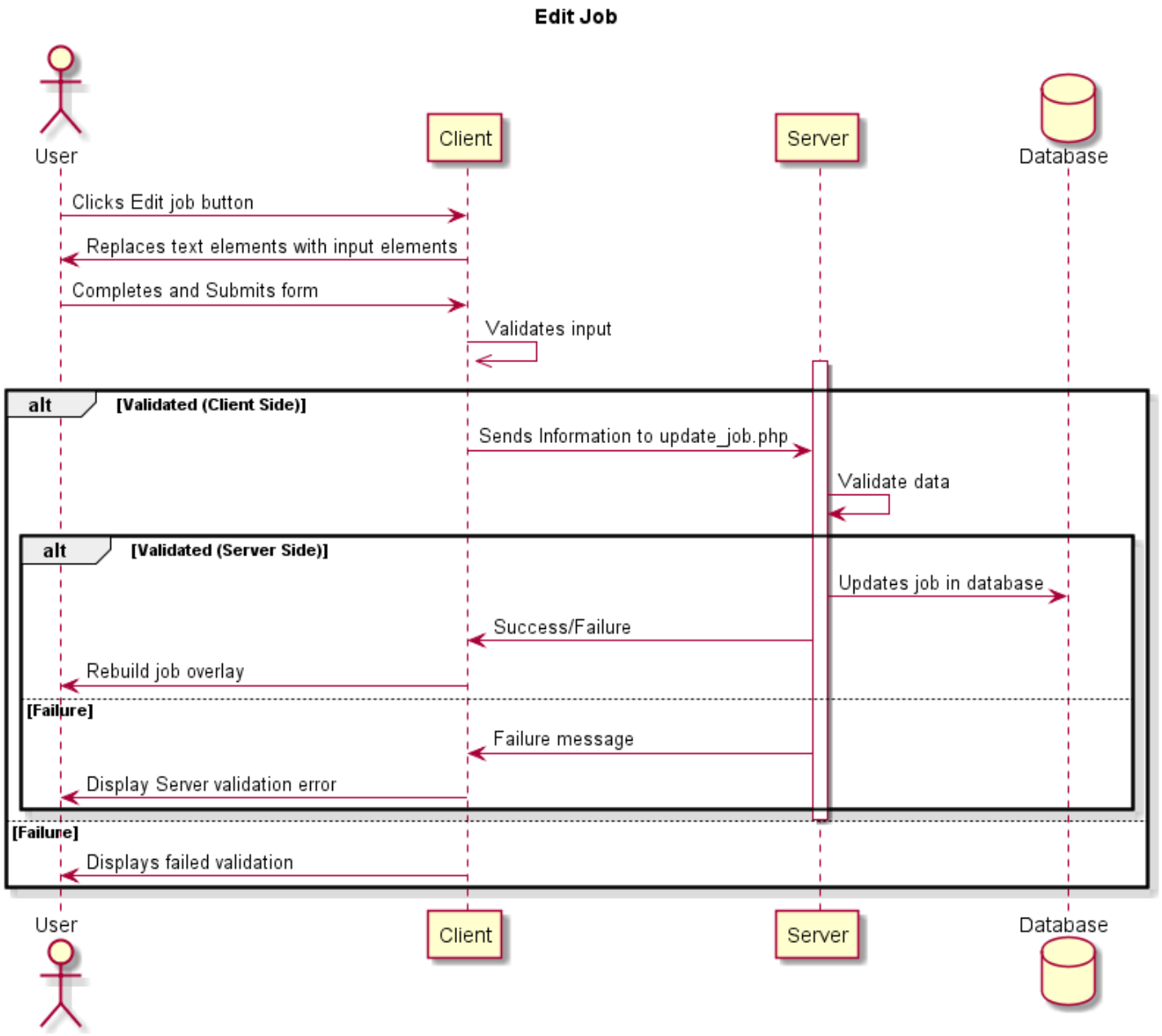
Extensions:

5.1. Validation errors

5.1.1. If the system encounters validation errors during the update process, such as invalid information format or conflicting data, it notifies the user and provides details about the encountered issues.

6.1. Database update errors

6.1.1. If the system encounters errors while updating the employee's details in the database, such as database connectivity issues or conflicts with existing data, it notifies the user and suggests retrying the update later.



When a user clicks the edit button *job-details/job-info.js-enableEditJob()* replaces all text elements with inputs of different kinds and adds event listeners to dynamically validate inputs.

When the user saves their changes *saveEditJob()* calls *job-details/update_job.php* with an ajax function. This document validates the inputs once again to make sure required fields are filled and that the length of specific fields are correct. It then sends the data to the database with an update statement. Once this is completed *saveEditJob()* turns the inputs back into text.

13. Delete Employee

Primary Actor: Construction Manager

Description: This function allows the manager to delete an employee.

Pre-condition: The user must be logged into the system (Use Case Five) and have the employee overlay open.

Post-condition: The employee is deleted.

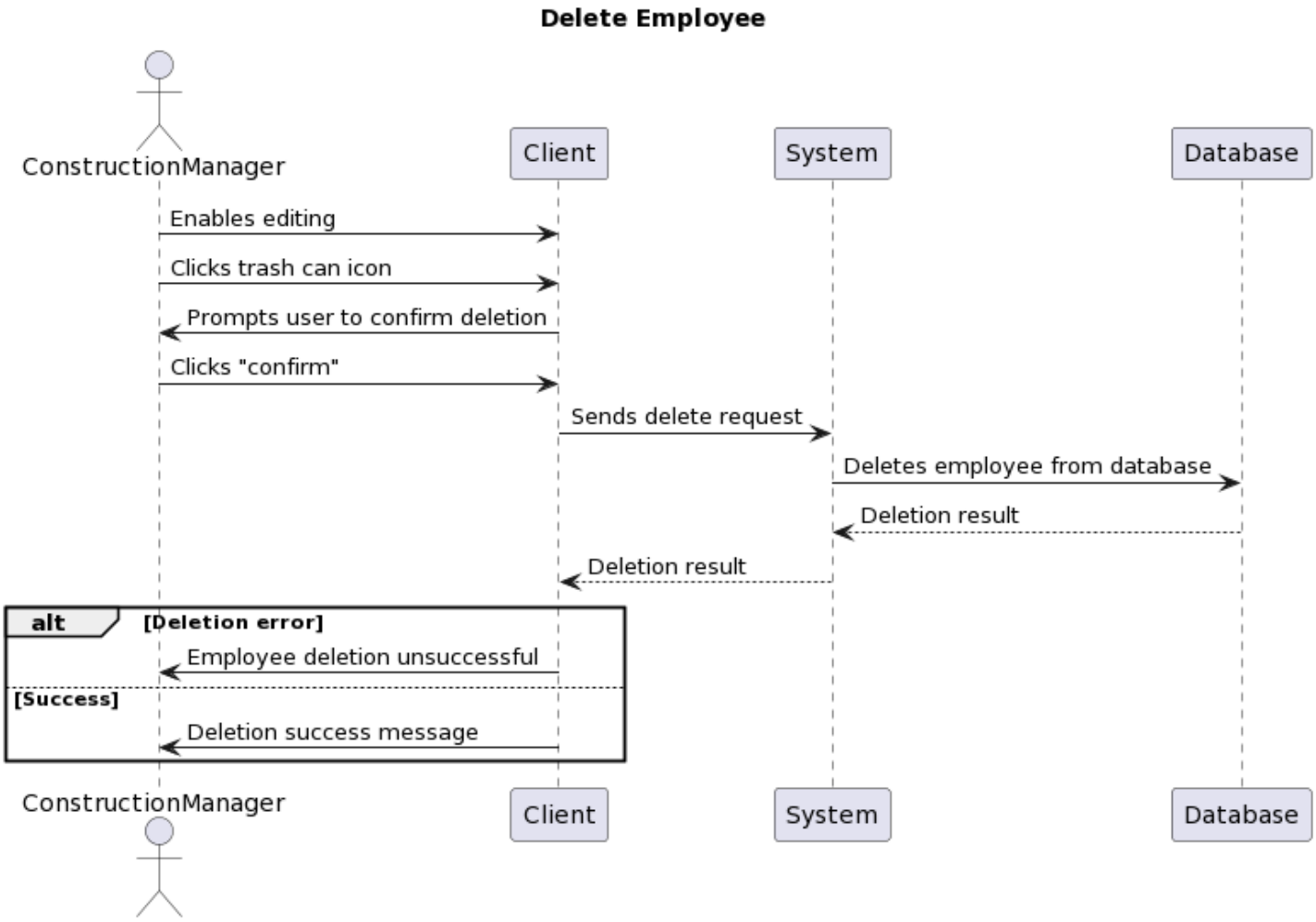
Main Scenario:

1. The user enables editing.
2. The user clicks the trash can icon.
3. The system prompts the user whether or not they want to delete the employee.
4. The user clicks "confirm."
5. The employee is deleted.

Extensions:

5.1 The employee does not exist.

5.1.1 The system sends an error message to the user.



The trash can icon on the employee overlay when editing is enabled sends an ajax request to *deleteEmployee.php* which deletes the employee from the database

14. Delete Job (**NEW**)

Primary Actor: Construction Manager

Description: The Delete Job function enables the Construction Manager to remove a job from the system.

Pre-condition: The Construction Manager must be logged into the system and have editing enabled for the job.

Post-condition: The selected job is deleted from the system if no employees are assigned to it.

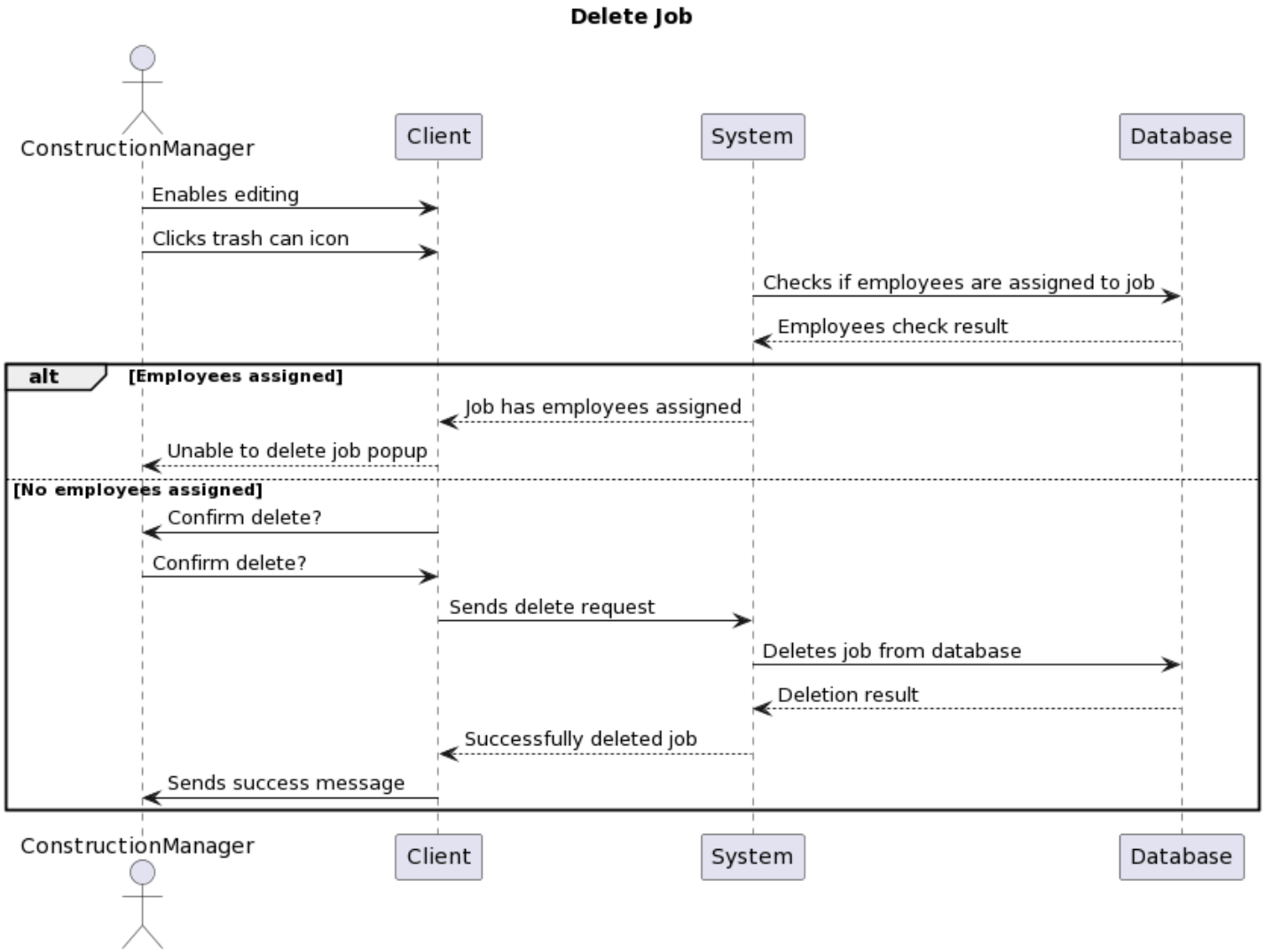
Main Scenario:

1. The Construction Manager enables editing for the job overlay.
2. The Construction Manager clicks on the trash can icon.
3. The system presents a confirmation prompt, asking if the Construction Manager wants to delete the job.
4. The Construction Manager clicks "confirm" to proceed with the deletion.
5. If no employees are assigned to the job, the system deletes the job from the system.

Extensions:

5.1 If the job does not exist:

5.1.1 The system sends an error message to the Construction Manager indicating deletion error.



15. View Employee Totals (Notable Feature) (**NEW**)

Refer to sequence diagram for use case 1

The employee totals bar on the bottom of the view job page contains the total number of employees for the current time as well as the total of the projected number of employees for the future months. The totals number is generated by *generate_totals.php* which is called by the javascript function *ajax_totals* which is called after an employee is moved, assigned, or when a projection number is changed. The *generate_totals.php* file generates the HTML required for the totals bar on the job page and the *ajax_totals* function inserts the generated HTML into the relevant element.

16. Auto Refresh (Notable Feature) (**NEW**)

Primary actor: Construction Manager

Description: The system will automatically refresh the page on a given interval to keep data current. It does this by checking the last updated time of the relevant tables and comparing it against the last check performed.

Pre-condition: The user must be logged in to the system (use case 5), viewing either the Employee Page (Use case 25), Job Page (Use case 1) or Presenter View (Use Case 22) and have their accounts refresh interval enabled.

Post-condition: The page currently being viewed is updated with the most recent changes

Main scenario:

1. The user is viewing one of the applicable pages
2. The interval time finishes and checks for new updates
3. If new updates exist the page is repopulated with new changes

Extensions:

2.1. Database update errors

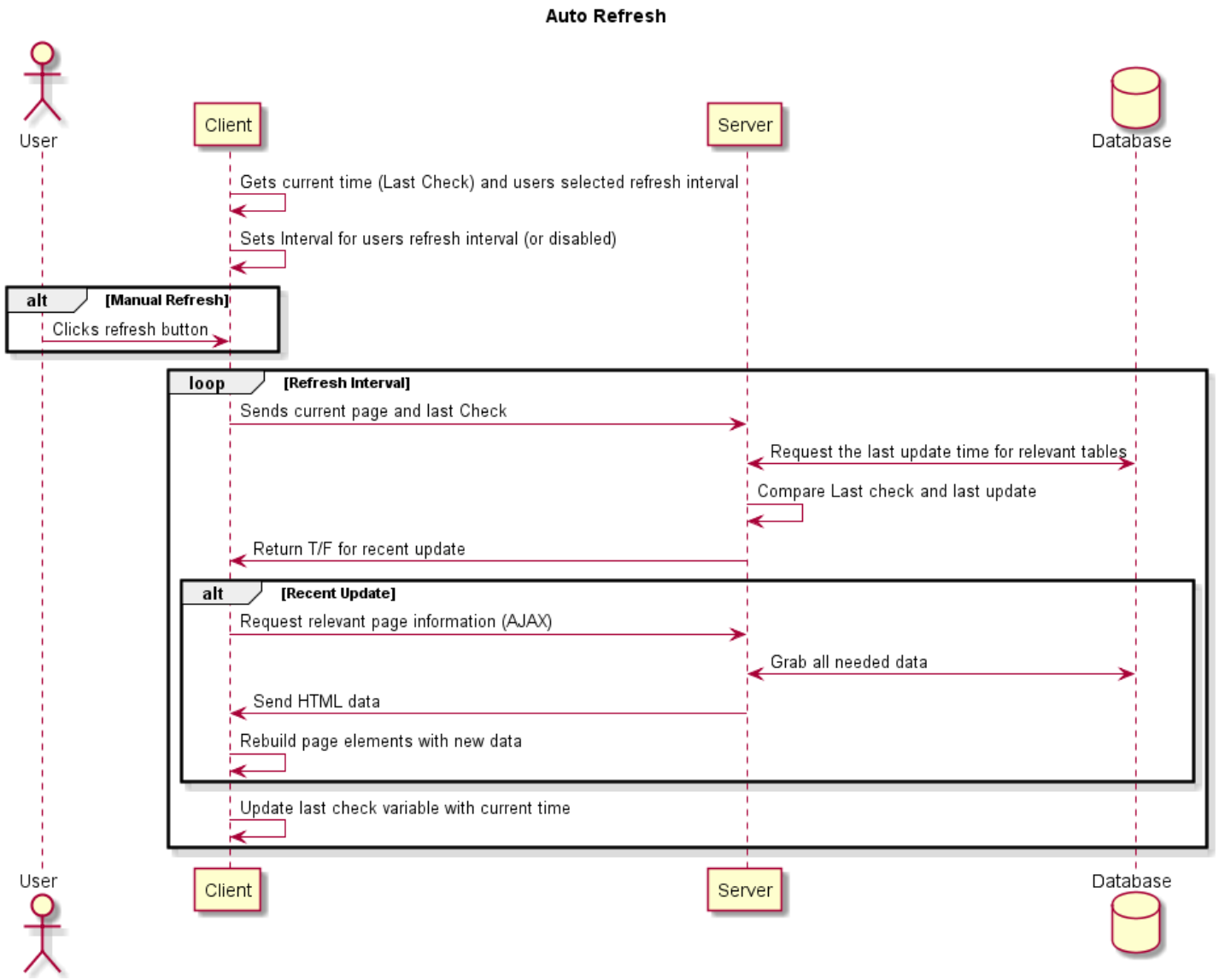
2.1.1. If the system encounters errors while fetching employee details in the database, such as database connectivity issues or conflicts with existing data, it notifies the user.

Auto refresh starts when either the *job-view.php* or *employees.php* page is loaded. These pages grab the current time in 'Seconds Since Epoch' and then *call src/refresh/refresh.js-refresh().* This function grabs the localstorage 'timer' item which is set on login and is set by the user on their account (Default is 30s), it then sets an interval for the *refresh.js-refreshHandler()* function. This function asynchronously calls *refresh/check-update.php* which queries the database for any updates after the current 'lastcheck' and returns true or false.

If there are recent updates for the relevant tables then *refreshHandler.js* will reload the necessary elements on the page.

Finally, lastcheck is then updated to the current time and the function repeats.

If the user clicks the button the interval is cleared and *refreshHandler()* is called as if the interval had just finished, once the function has completed the interval is set once again and the continues the countdown.



17. Duplicate Employee

Description:

Primary actor: Construction Manager

Description: Allows for quick duplication of employee assignment

Pre-condition: The user (Construction Manager) must be logged in to the system and accessing the job page.

Post-condition: The employee's assignment is duplicated to another job, saving time and effort for the user.

Main scenario:

The user selects an employee on a job and initiates dragging the employee to another job.

While dragging the employee, the user holds down the Ctrl key.

The system detects the Ctrl key press and switches the move function to the "Duplicate Employee" function.

The user drops the employee onto another job.

The system recognizes the "Duplicate Employee" function and creates a new assignment for the employee on the target job, replicating the details of the original assignment.

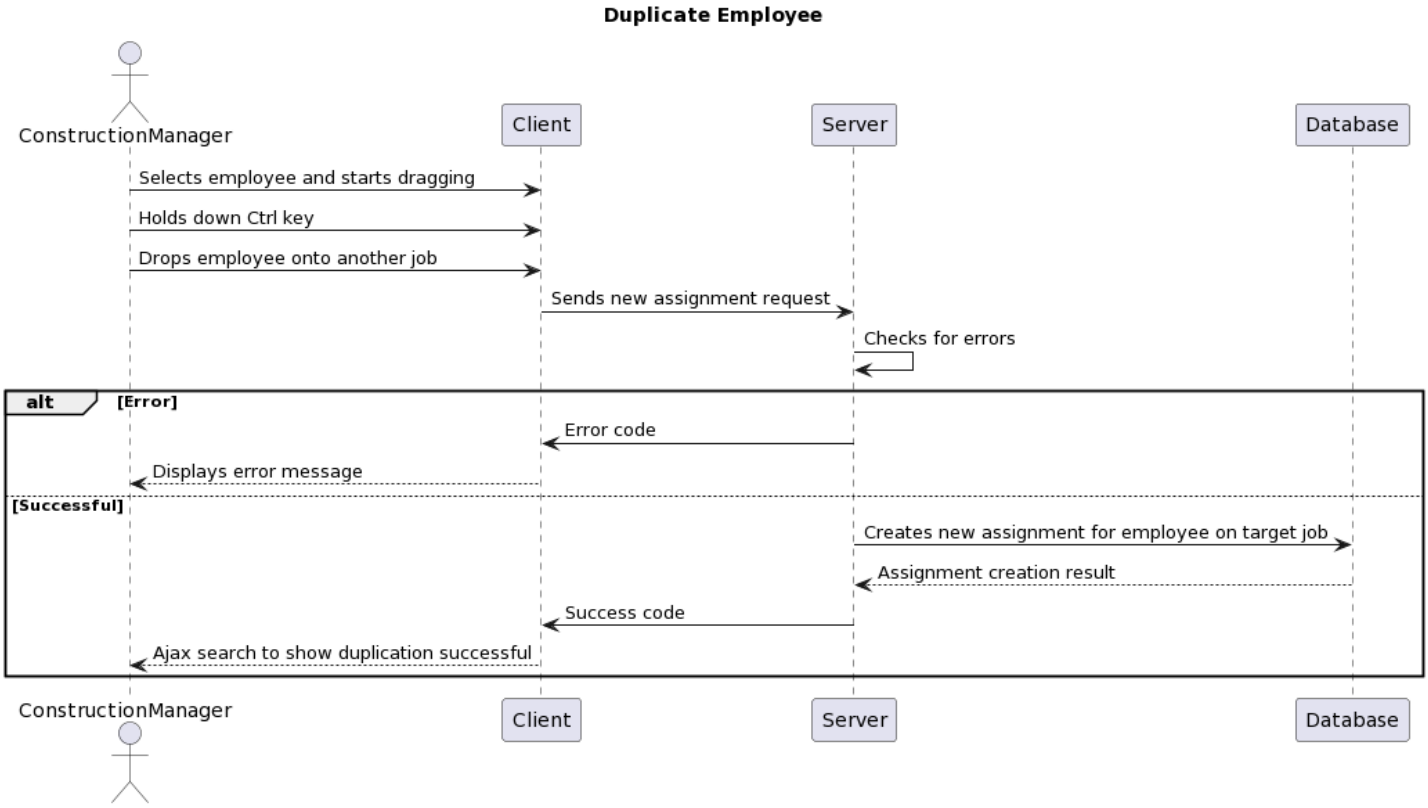
Extensions:

6.1. Ctrl key not held

6.1.1. If the user does not hold down the Ctrl key while dropping the employee, the system executes the standard move function and assigns the employee to the new job, removing them from the original job.

6.2. Error during duplication

6.2.1. If an error occurs during the duplication process, such as a database connection issue or a failed operation, the system notifies the user with an error message and provides information about the issue.



The duplicate employee function is an add on that is available for use on the job page. When a user is dragging an employee from one job to another, they have the option to hold down the Ctrl key when dropping the employee to change the move function to a new assignment function. This function allows for the user to not have to open the employee list if they want to duplicate an employee on multiple jobs.

18. Login/Admin Login

Primary actor: Construction Manager, Admin

Description: Allow users to log in to their own private account.

Pre-condition: User must have an account.

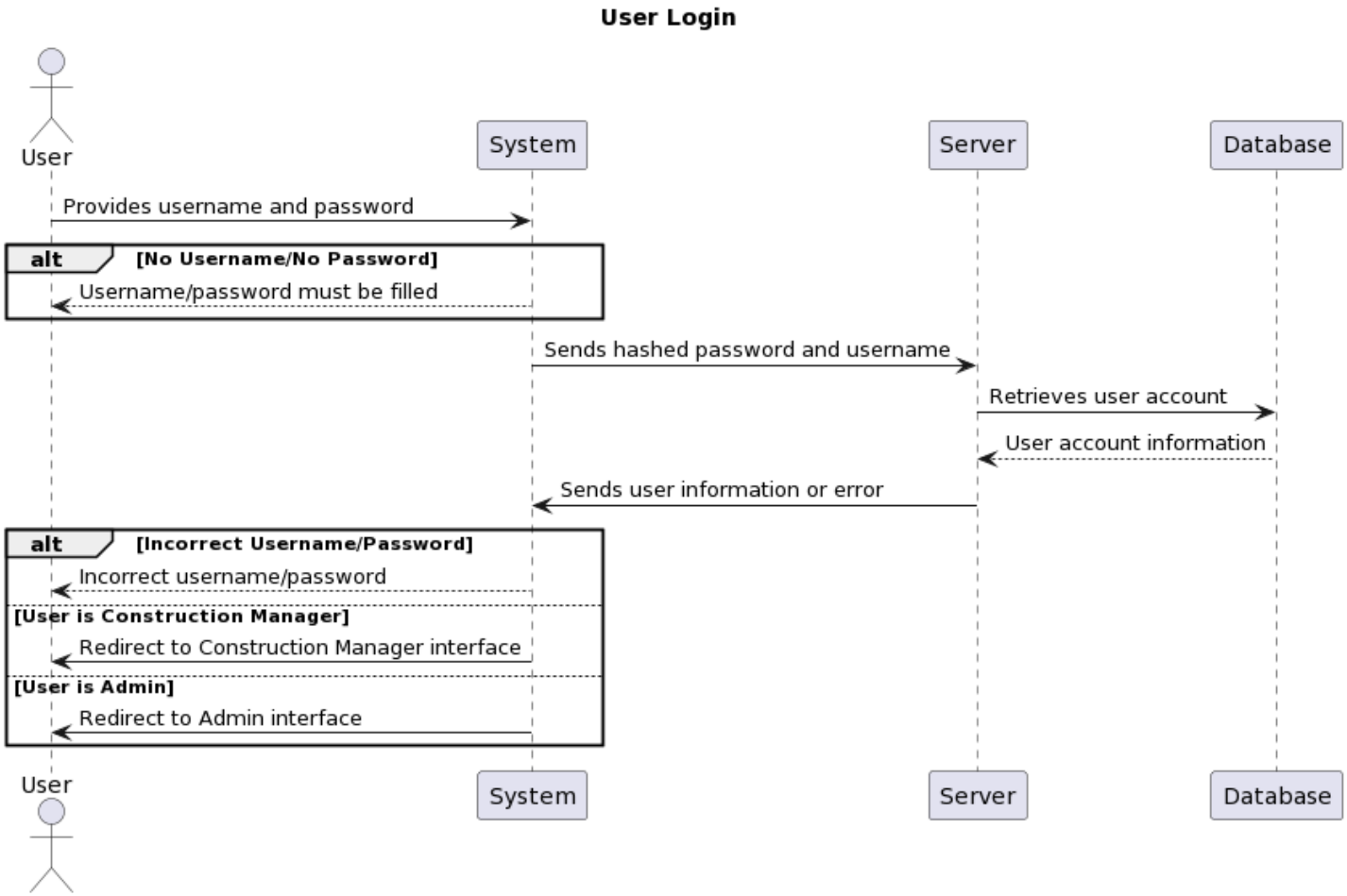
Post-condition: The user will be logged in to the system.

Main scenario:

1. User provides username and password.
2. User tells system to login.
3. System tells user is successfully logged in

Extensions:

1.1. No Username/No password
    1.1.1. The system tells the user that the username/password must be filled or is incorrect
2.
2.1. Server/database Timeout
    2.1.1. System tells user the server/database has timed out.
2.2. Incorrect password/username
    2.2.1. System informs user of incorrect username/password



The login page (*login.php*) contains a simple login form. This login form, when submitted, sends the inputted information to *processlogin.php* which queries the database with the hashed password and username and then returns the user information if a user is found. Depending on what role this user is in the database (admin, regular user, or projector user) the user is redirected to the appropriate page. If a user is not found, the script redirects back to *login.php* with a flag that indicates that the login was unsuccessful

19. Logout

Primary actor: Construction Manager/Admin

Description: Allow users to log out from their private account.

Pre-condition: User must be logged in to the system.

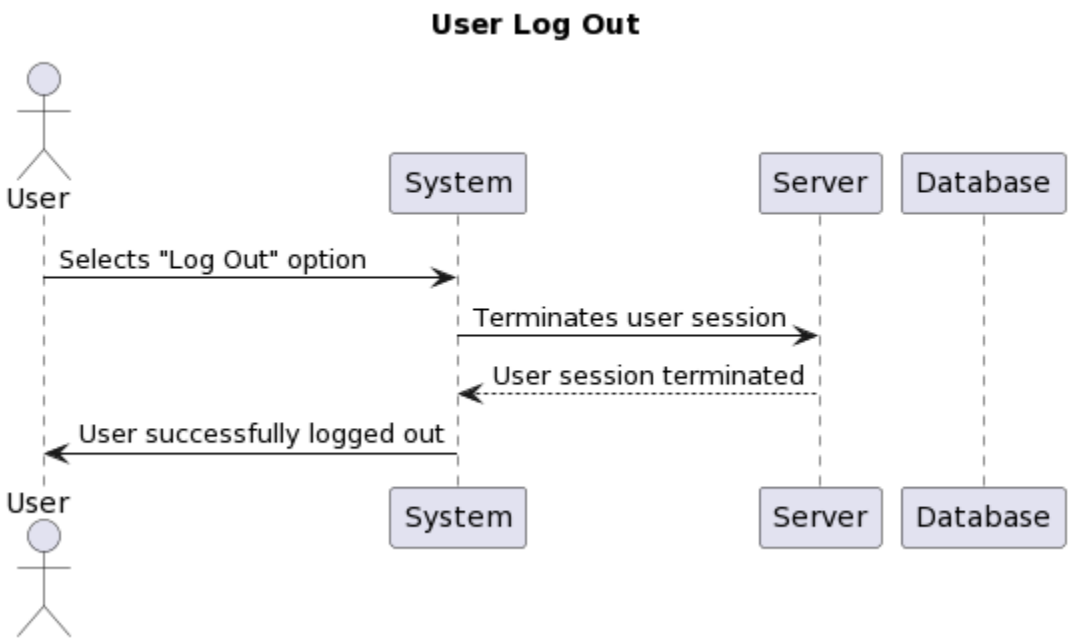
Post-condition: The user will be logged out of the system.

Main scenario:

1. User selects the "Log Out" option from the portal.
2. The system terminates the user's session and logs the user out.

Extensions:

None.



Important Use Cases

20. View Account Page (**NEW**)

Description:

Primary actor: Construction Manager/Admin

Description: This use case involves allowing users to view their account details and access functions that are on the account page. The purpose is to provide users with an interface to manage their account information and perform actions that are associated with their account.

Pre-condition: User must be logged in to the system.

Post-condition: The user views their account page and accesses the hardcoded functions.

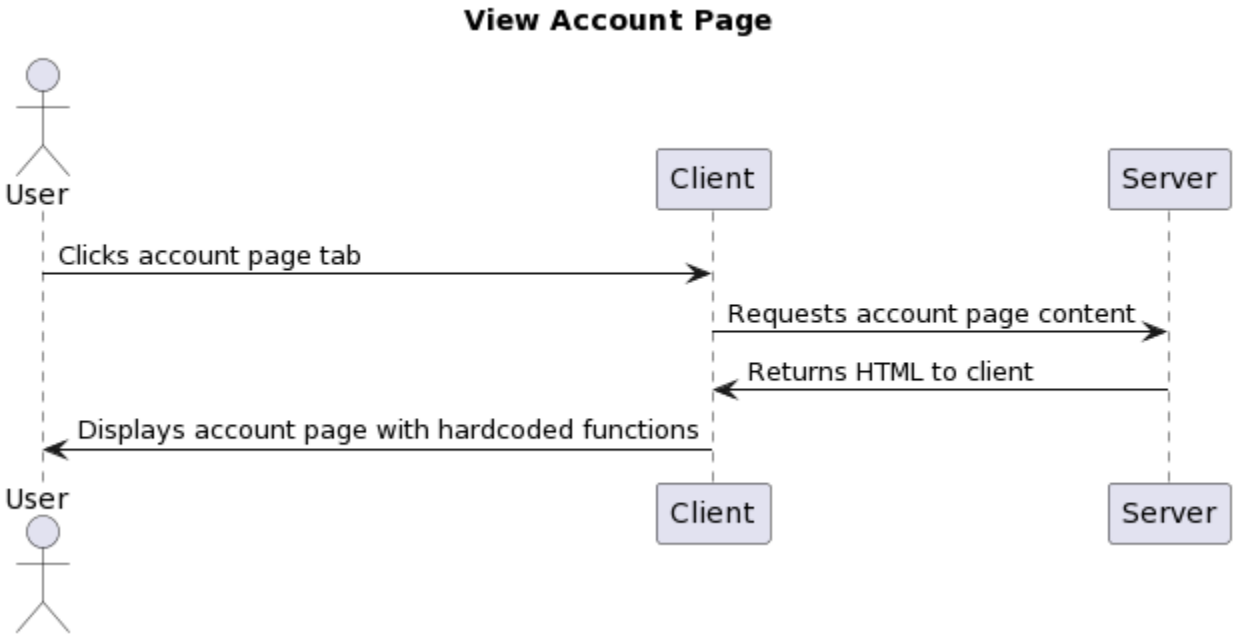
Main scenario:

User clicks the account page tab in the navigation bar

The account page loads.

Extensions:

None.



21. Change Password

Primary Actor: Construction Manager

Description: The Change Password function allows the user to change their password within the system.

Pre-condition: User must be logged in to the system.

Post-condition: The user's password is successfully changed, and the system updates its records accordingly.

Main Scenario:

1. User decides to change their password.
2. User navigates to the "Change Password" option on the account tab.
3. System prompts the user to enter their current password and new password.
4. User enters their current password and new password.
5. System verifies that the current password is correct and updates the user's password to the new one.
6. System confirms the successful password change to the user.

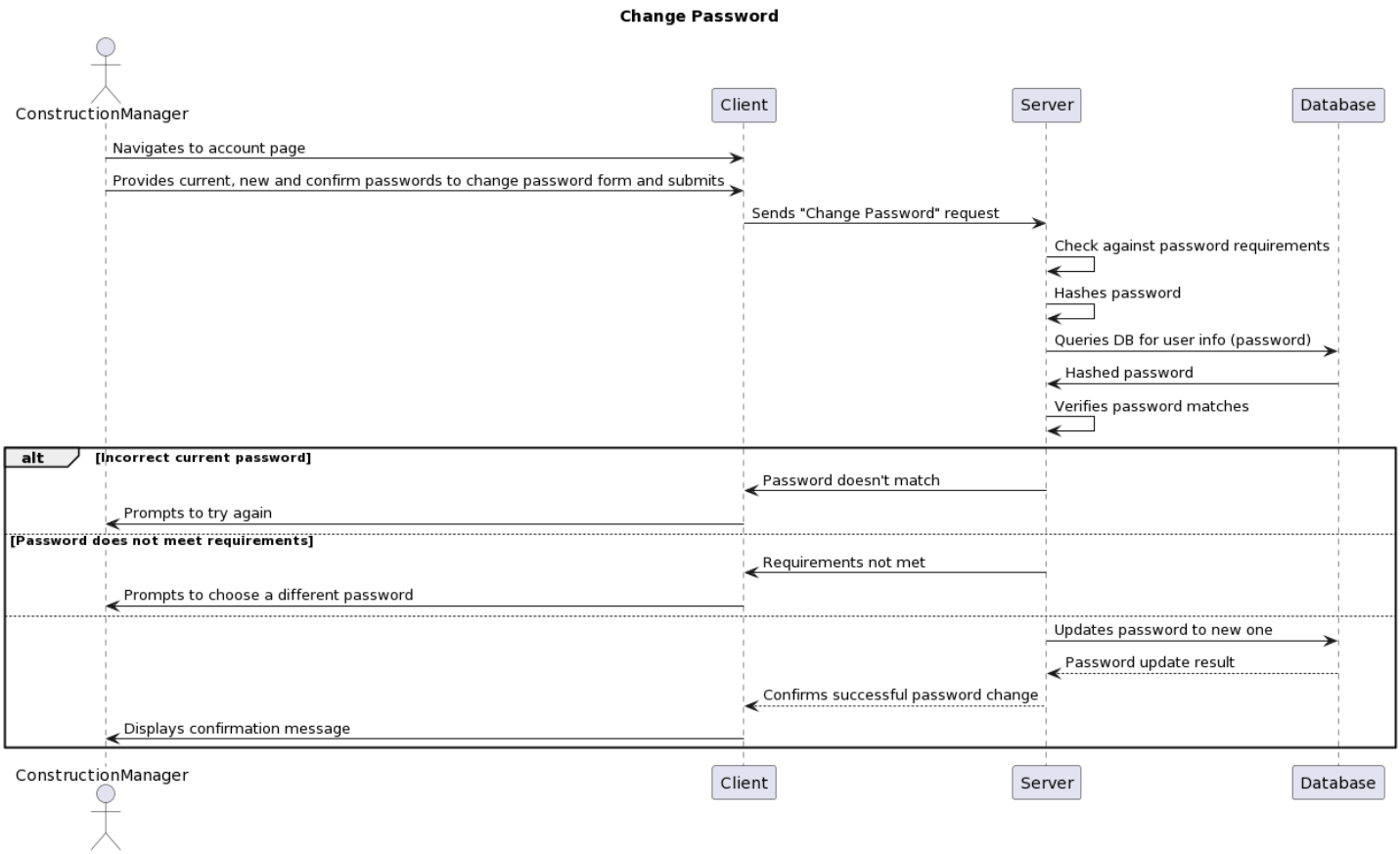
Extensions:

5.1 Incorrect current password

5.1.1 The system determines that the entered current password is incorrect and prompts the user to try again.

5.2 Password does not meet requirements

5.2.1 The system detects that the password does not meet complexity requirements and prompts the user to choose a different password



The change password function is a form that is available on *account.php*. When the form is submitted, the page sends an ajax request to *change_password.php* which does accuracy, availability, and complexity checks on the password before changing the password in the database. If the checks do not pass, a message is sent back from *change_password.php* to the client indicating what the user should change.

22. View Presenter View

Primary actor: Construction Manager

Description: Allow users to view a view-only schedule portal

Pre-condition: User must have a server to host the website locally.

Post-condition: The user can view the simplified schedule/projector view.

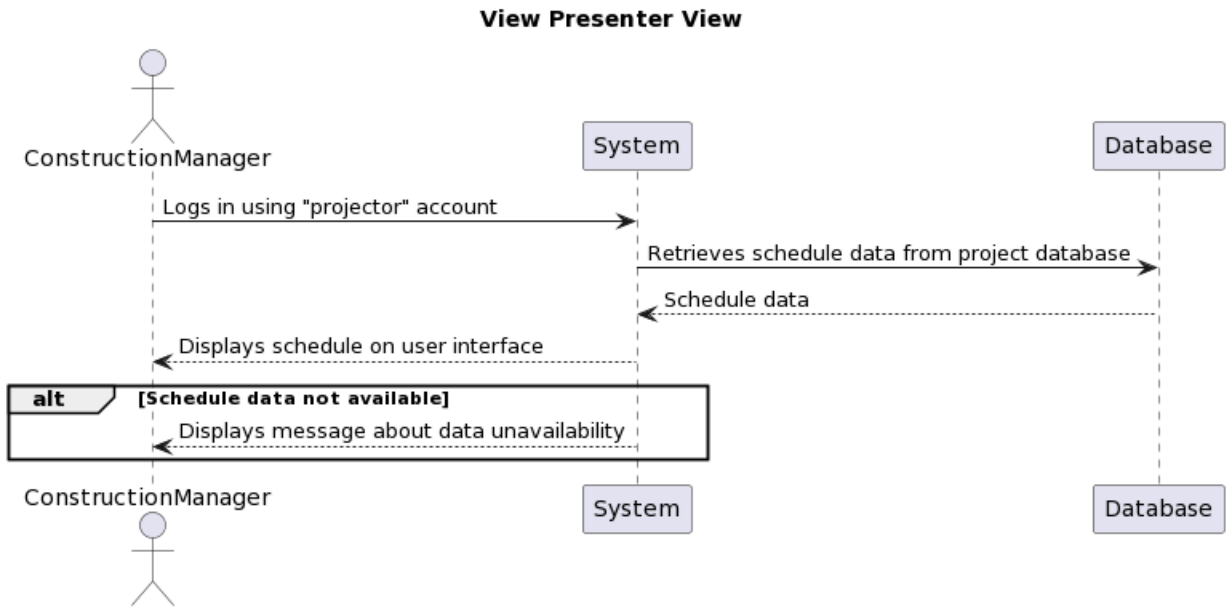
Main scenario:

1. User logs in using the “projector” account
2. The system retrieves the schedule data from the project database.
3. The system displays the schedule on the user interface, presenting tasks, employees, start and end dates, and other relevant details.

Extensions:

2.1. Schedule data not available

2.1.1. If there is no schedule data available in the project database, the user interface displays a message informing the user of the fact



Viewing the projector view is a extra function of the login process flow (use case #18) where if the login script detects that the user logging in is of the “projector” type, then it redirects directly to the projector view and does not set any of the other variables that would normally be set with a regular user login (such as user ID and permissions)

23. View Admin Functions (**NEW**)

Primary actor: Admin Manager

Description: Allow admin user to access admin-only functions

Pre-condition: User must have an admin user account and login

Post-condition: The user can view admin-only functions

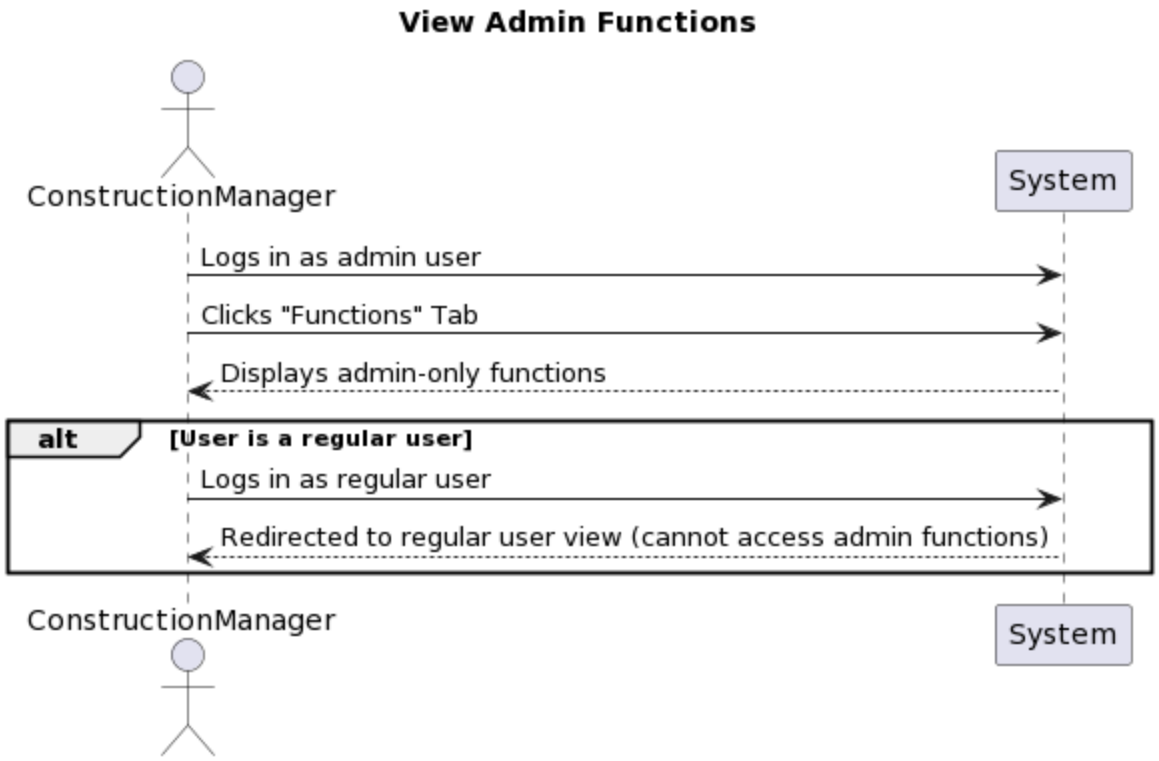
Main scenario:

1. User logs in as an admin user
2. User click the “Functions” Tab
3. The system displays the admin-only functions

Extensions:

1.1. The user is a regular user (non-admin user)

2.1.1. The system won’t display the “Functions” Tab



Admin functions is a page that is available if the user is logged in as an administrative account. Each 'admin' page has a check that checks the *$_SESSION('is_admin')* variable before loading the page to ensure that the user has permissions to view the page. The tab for the admin page is not available if the user is not logged in as an admin as well.

24. Create Manager Account

Primary actor: admin

Description: Allow admin to create a new account in the system.

Pre-condition: admin must already log in to create an account

Post-condition: a new user's account is successfully created in the system.

Main scenario:

1. Admin navigates to the functions tab.
2. Admin enters a username and presses create account.
3. The system checks for username availability and uniqueness.
4. The system creates a new account
5. The system securely stores the account information.
6. The system provides feedback to the user indicating the successful creation of the account as well as a temporary password which the admin can then pass on to the manager.

Extensions:

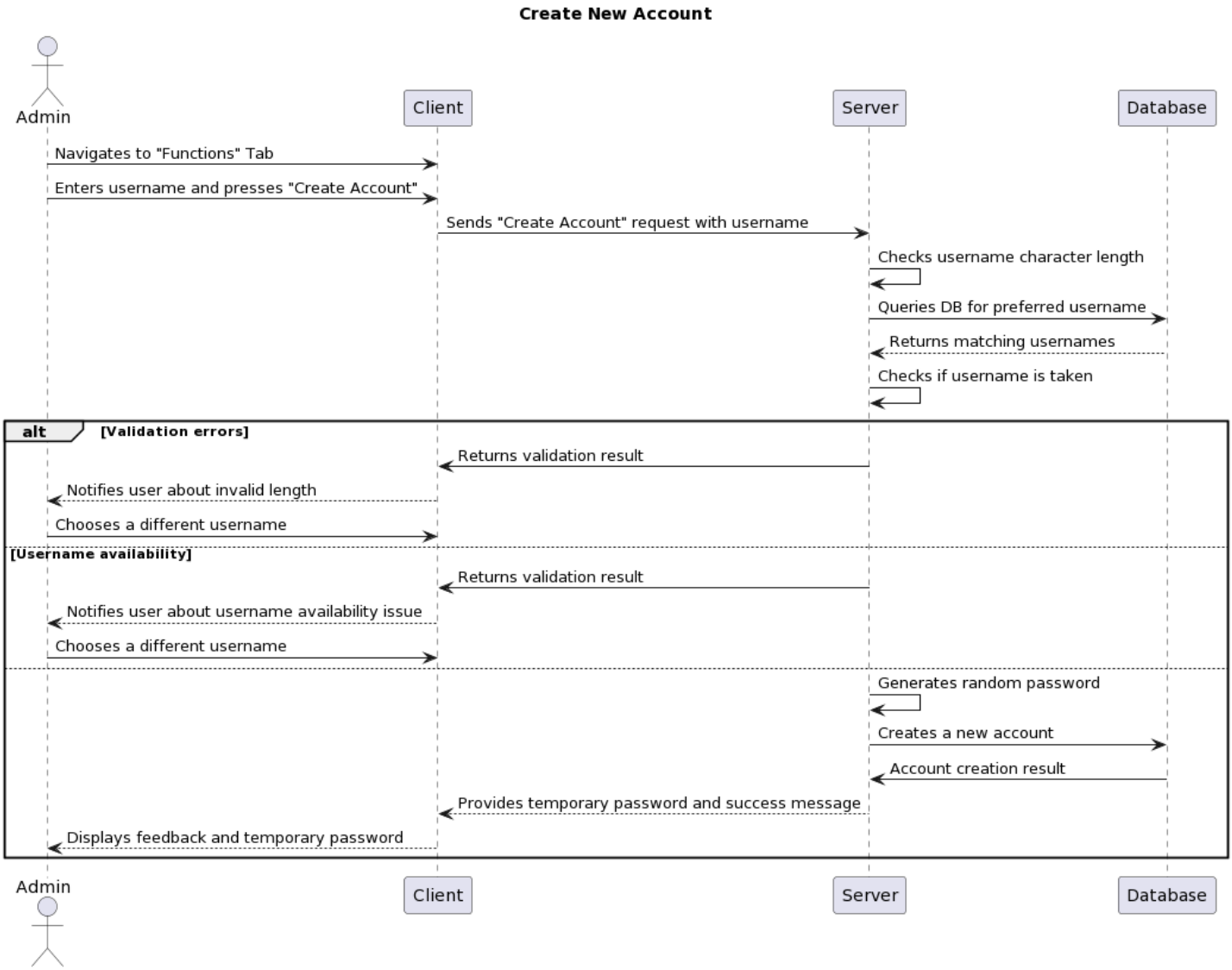
4.1 Validation errors

4.1.1 If the system encounters validation errors during the registration process, such as missing required fields or conflicting data, it notifies the user and provides details about the encountered issues.

4.1.2 The system prompts the user to correct the errors before resubmitting the form.

5.1 Username availability

4.1.3 If the chosen username is already taken or fails to meet the availability criteria, the system notifies the user and prompts them to choose a different username.



Create manager account is a function available on *admin_functions.php* as a form. When the user enters a username and submits the form, the data is sent via ajax request to *create_manager.php* which processes the data, checks that the username is not taken or invalid, and creates the user account with a temporary generated password. After the account is created, the temporary password and success message is passed back to the client which then displays a success message pop up with the temporary password available to be copied.

25. View Employee Page (**NEW**)

Description:

Primary actor: Construction Manager

Description: This use case involves the Construction Manager logging into the system and accessing the "View Employee Page" by navigating to the respective section. The purpose is to provide the Construction Manager with a overview of employees.

Pre-condition: The Construction Manager must be logged in to the system.

Post-condition: The Construction Manager is presented with the Employee Page, which displays all employees.

Main scenario:

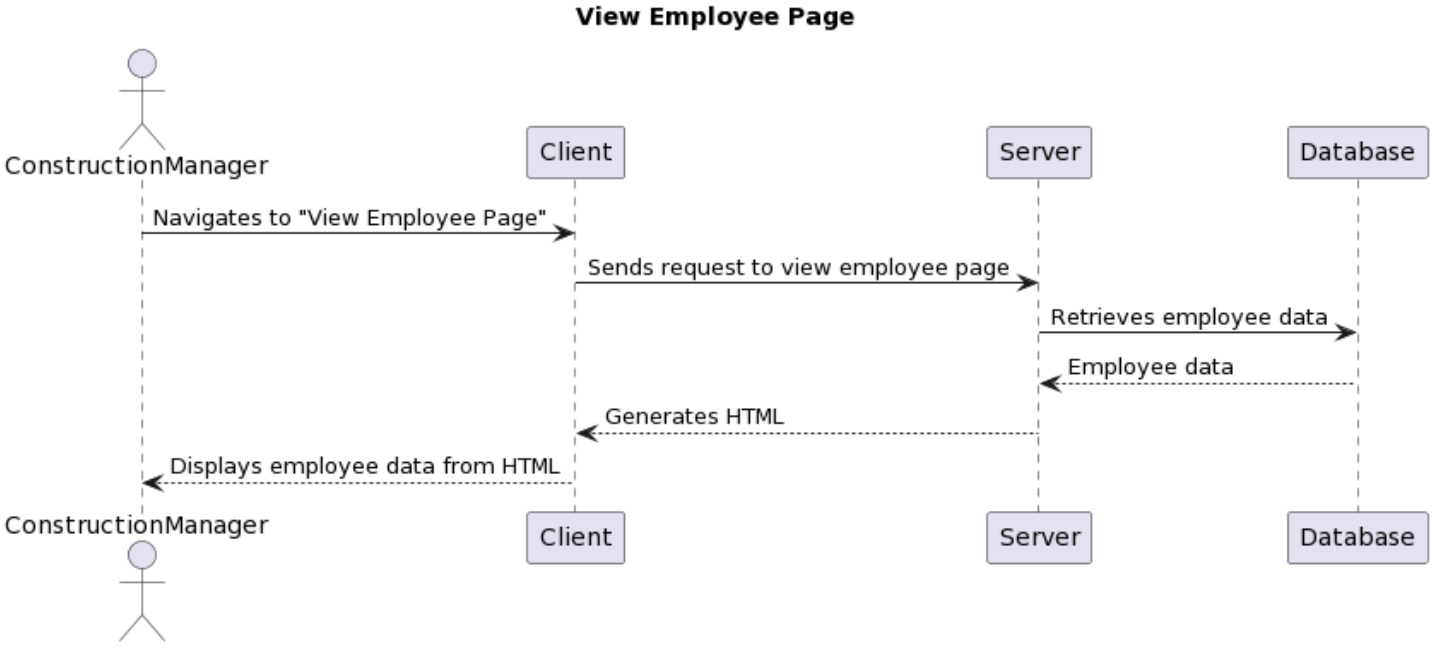
The Construction Manager navigates to the "View Employee Page" section.

The system fetches and displays employee-related information on the "View Employee Page."

Extensions:

3.1 Error Display

3.1.1 If there are errors or issues with fetching and displaying the employee-related information, the system visually shows the Construction Manager an error message, indicating the nature of the problem.



The employee page is a file called *employees.php* which can be accessed by a logged in user either through the navigation bar or by directly typing in the file path. *employees.php* makes ajax requests to *generate_list.php* to generate the necessary HTML code for the employee sections.

26. Archive Job

Primary actor: Construction Manager

Description: Allow the user to remove jobs from the active panel

Pre-condition: The user must be logged in to the system.

Post-condition: The job is removed from the main page and added to the archive page, employees assigned to the job are added to the unassigned tab

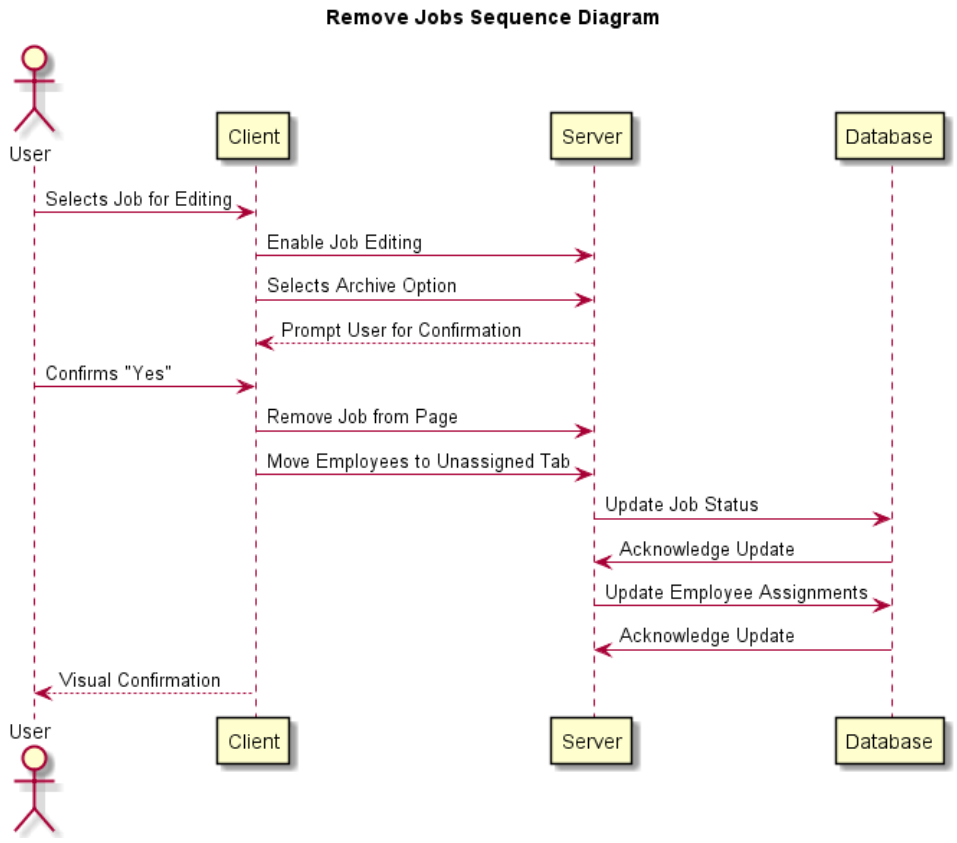
Main scenario:

1. User selects a job and enables editing
2. User selects the archive option for the job
3. System prompts the user if they are sure
4. If the user selects yes, the system visually removes the job from the page
5. Employees are visually moved to the unassigned tab
6. The system moves the job to the archived table

Extensions:

7.1. Database error

7.1.1. Database fails to update with new data due to connection issues or a failed database constraint. The user is notified of this.



After selecting a job and enabling editing, the option to archive the job appears. When the manager confirms, the system visually removes the job from the job view, while assigned employees are shifted to an unassigned status. The job's status is updated in the database, and the employees' assignments are modified. The user receives a visual confirmation of the successful removal. In case of database issues, the system notifies the manager with a failure/warning message.

27. Unarchive Job (**NEW**) Use Case: Unarchive Job

Primary Actor: Construction Manager

Description: Enables the user to unarchive a previously archived job, restoring it to the active panel, and reassigning employees as needed.

Pre-condition: The user must be logged in to the system.

Post-condition: The archived job is successfully restored to the active panel

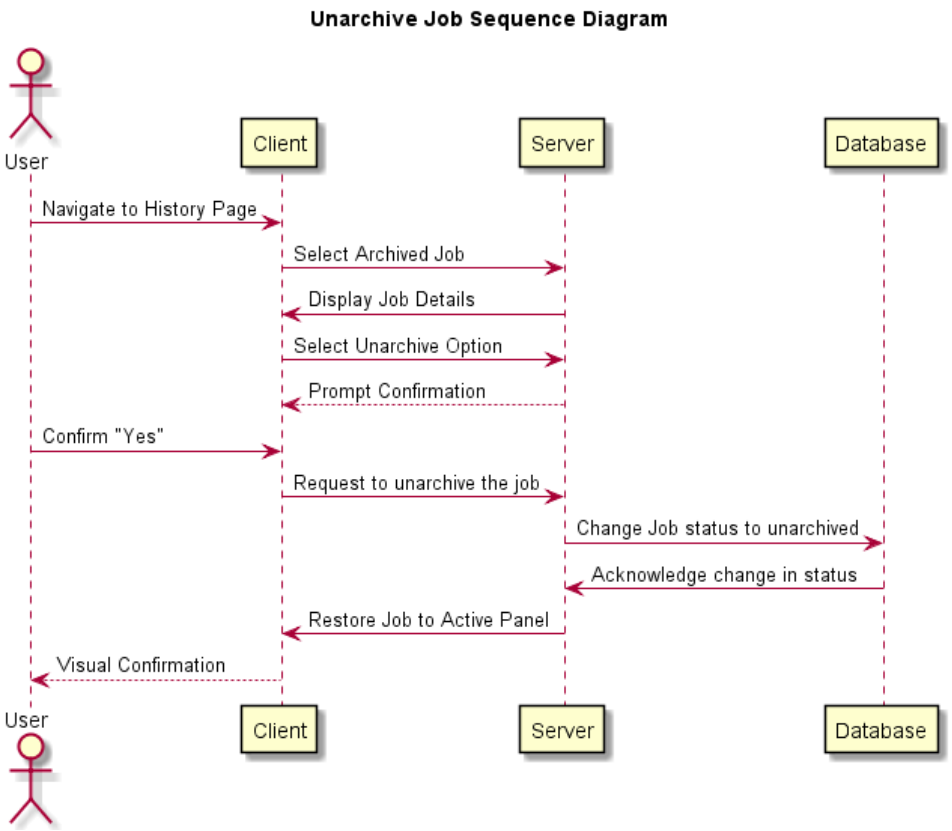
Main Scenario:

1. User navigates to the history page where archived jobs are listed.
2. User selects an archived job for unarchiving.
3. System displays details of the selected archived job.
4. User selects the option to unarchive the job.
5. System prompts the Construction Manager for confirmation.
6. User confirms "Yes" to proceed with unarchiving.
7. System restores the job to the active panel and visually updates the user interface.
8. User receives a visual confirmation of the successful unarchiving.

Extensions:

9.1. Database Error:

If there is a database error during the unarchiving process, the system notifies the User.



28. Show Archived Employees (**NEW**)

29. Archive Employee (**NEW**)

Primary actor: Construction Manager

Description: Allow the user to remove employees permanently from the job page (archive)

Pre-condition: The user must be logged in to the system.

Post-condition: The employee only shows in the archive not in the active job page.

Main scenario:

1. User enters the edit page
2. User selects the edit option for an employee
3. User changes the employee status to archived
4. User clicks save icon
5. The System prompts the user if they are sure
6. Employee is removed from active status and all jobs and hidden from view on employee page
7. The system sets the employee to archived in the database

Extensions:

7.1. Database error

7.1.1. Database fails to update with new data due to connection issues or a failed database constraint.

30. Unarchive Employee (**NEW**)

31. Load Database Backup

Primary Actor: Construction Manager

Description: The Load Database Backup function allows the user to load a backup of the database from a .sql file.

Pre-condition: User must be logged in to the system and they must have an existing .sql backup file

Post-condition: The database is successfully restored from the backup .sql file.

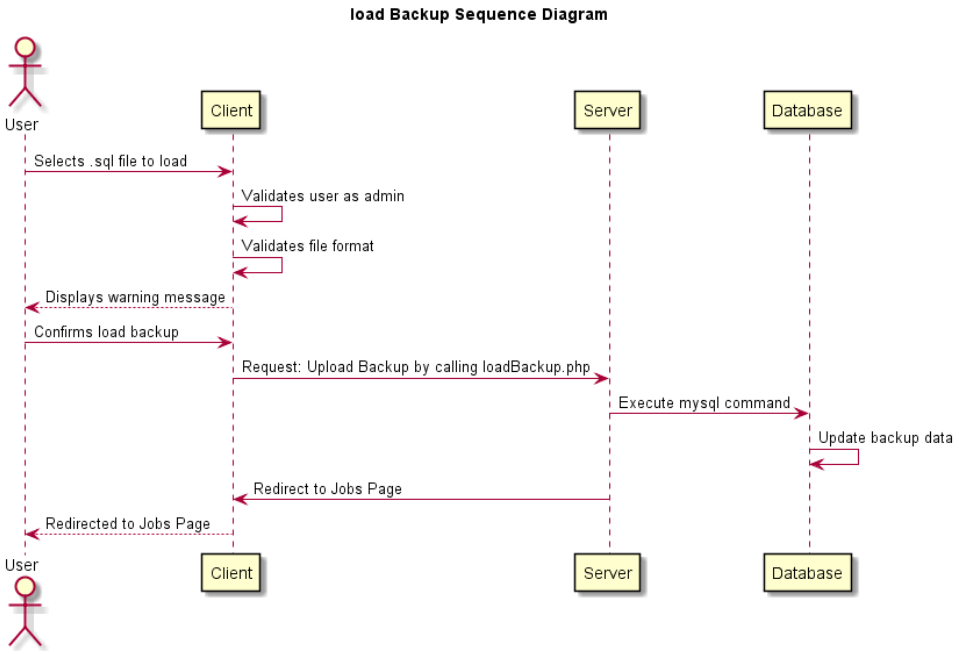
Main Scenario:

1. User decides to load a database backup.
2. User navigates to the “functions” tab within the system’s interface and selects the “Load Database Backup” option.
3. System prompts the user to select the backup .sql file.
4. User selects the backup .sql file.
5. System loads the data from the .sql file and restores the database to its state at the time of the backup.
6. System confirms the successful database restoration to the user.

Extensions:

4.1 Invalid .sql file

4.1.1 The system determines that the selected .sql file is invalid and prompts the user with an error message.



To load a backup file, the user must be logged in as an admin first, then that user can select a backup file to load from the dropdown menu. After a file is selected, the user then clicks on the upload button which triggers a client-side warning message. Upon user confirmation, the client side calls *loadBackup.php*, which executes a mysql command through web-server container's command line. After a successful load, the user is prompted a message and redirected to the jobs page.

32. Reset User's Password

Primary Actor: Admin

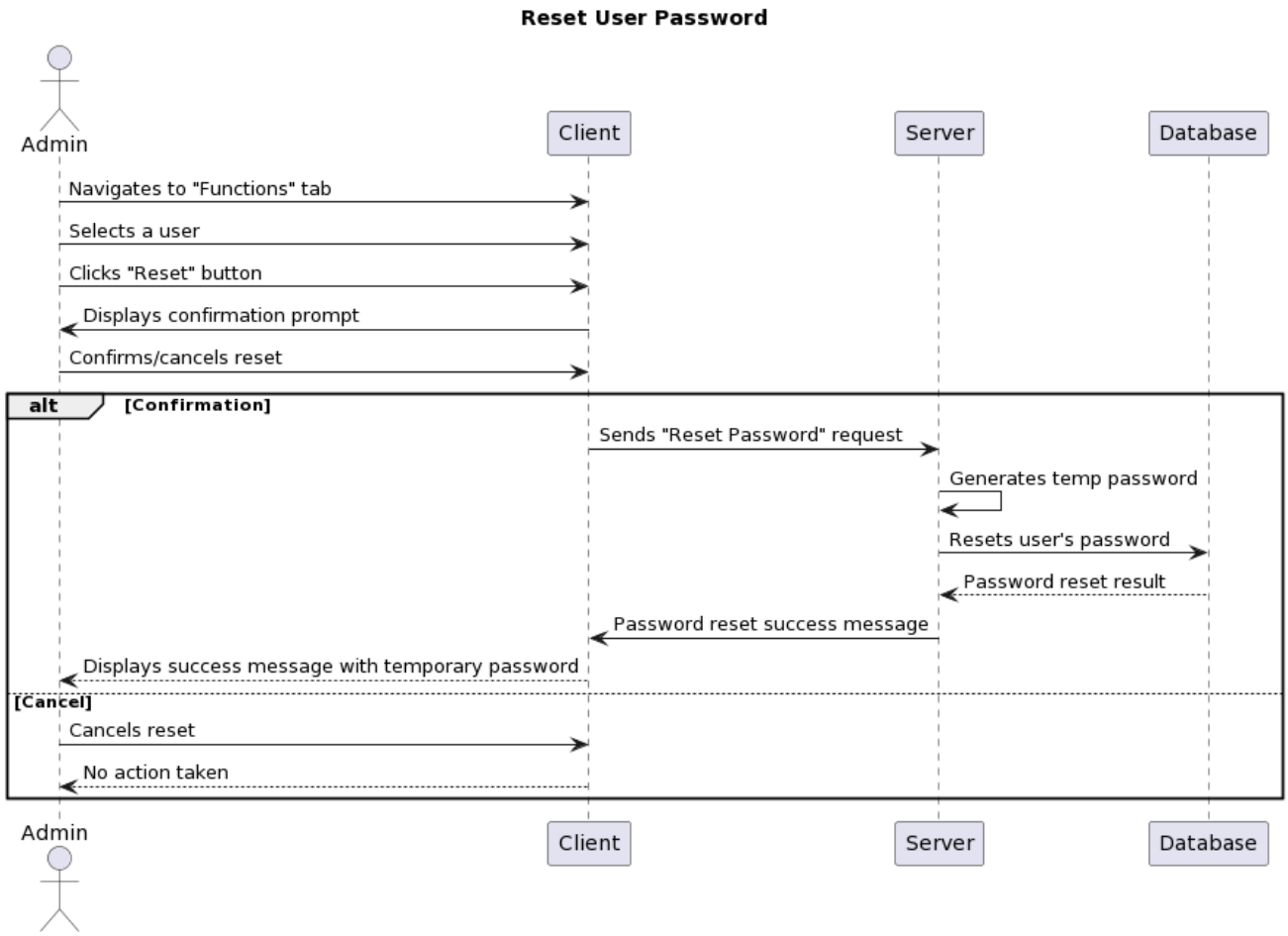
Description: The Reset User Password function allows the admin to reset the password of a user within the system.

Pre-condition: Admin must be logged in to the system.

Post-condition: The selected user's password is successfully reset, and the system updates its records accordingly.

Main Scenario:

1. Admin decides to reset a user's password.
2. Admin navigates to the "Functions" tab within the system's interface and selects a user.
3. Admin clicks the "Reset" button next to the selected user's profile.
4. Admin confirms that they want to reset the user's password
5. System resets the selected user's password to a temporary value.
6. System confirms the successful password reset to the admin and displays temporary password.



The reset user password function is a form that is available on the administrative function page *admin_functions.php*. When the admin user clicks the reset password with a user selected and confirms the reset, the system sends a request to the server to reset the password via an ajax request to *reset_password.php*. After the password is reset by *reset_password.php*, a confirmation message and a temporary password is provided to the administrative user via a popup.

Nice To Haves

33. Sort/Filter/Search Employees (**NEW**)

Primary actor: Construction Manager

Description: Allow the user to reorder employee/find specific employee

Pre-condition: The user must be logged in to the system.

Post-condition: The employees are displayed in the users desired order/the specific employees are displayed.

Main scenario1 (sort/filter):

1. User selects the desired order preference for employees.
2. The employee list is refreshed and displayed as their desired order preference.

Main scenario 2 (search):

1. User types in the employee name they want to search in search bar.
2. The employee(s) that match the searching criteria is(are) displayed at their current display order

Extensions:

1.1 Entered employee names don't exist.
    1.1.1 No employee shown on screen.
1.2 The employee database is empty.
    1.2.1 Display "no data" error message.

34. Sort/Filter/Search Jobs (**NEW**)

Primary actor: Construction Manager

Description: Allow the user to reorder jobs/find specific jobs

Pre-condition: The user must be logged in to the system.

Post-condition: The jobs are displayed in the users desired order/the specific jobs are displayed.

Main scenario1 (sort/filter):

1. User selects the desired order preference for jobs.
2. The job list is refreshed and displayed as their desired order preference.

Main scenario 2 (search):

1. User types in the job name they want to search in search bar.
2. The job(s) that match the searching criteria is(are) displayed at their current display order

Extensions:

1.1Entered job names don't exist.

1.1.1No job shown on screen

1.2The job database is empty.

1.2.1Display "no data" error message

35. Upload Employee Image (**NEW**)

Primary actor: Construction Manager

Description: Allow the user to upload images to employee profiles.

Pre-condition: The user must be logged in to the system.

Post-condition: The new image is uploaded to the server and the database is updated with this new photo. The visual representation on the Employee Overlay (Use case 8) and the Job view (Use case 1)/Employee View (Use case 25) should be updated with this new photo.

Main scenario:

8. User opens the employee details (Use case 8)
9. They click on the employees image to open the file browser.
10. They select a photo to use.

11. It is validated, compressed and downloaded to the database.
12. The employees instances on the page are updated.

Extensions:

11.1. Database error

11.1.1. Database fails to update with new data due to connection issues or a failed database constraint.

11.2. Validation error

11.2.1. The image type is not correct and the user is notified

11.2.2 The image is too large and is rejected by the filesystem

36. Custom Sort Jobs [+ upload to projector] (**NEW**)

Description:

Primary Actor: Construction Manager

Description: The Custom Order function allows the Construction Manager to customize the order in which jobs are displayed on the page. The user sets a custom sort option, retrieves jobs based on the custom order, and can then rearrange the job titles using an overlay. After arranging the jobs, the user can save the custom order, either to their personal account or the projector account.

Pre-condition: The user (Construction Manager) must be logged in to the system.

Post-condition: The jobs' order is customized and saved according to the user's preferences.

Main Scenario:

User navigates to the job page.

User selects the "Custom Order" option.

The page generates an overlay with draggable job titles.

User rearranges job titles by dragging and dropping.

User clicks "Save" or "Upload to Projector."

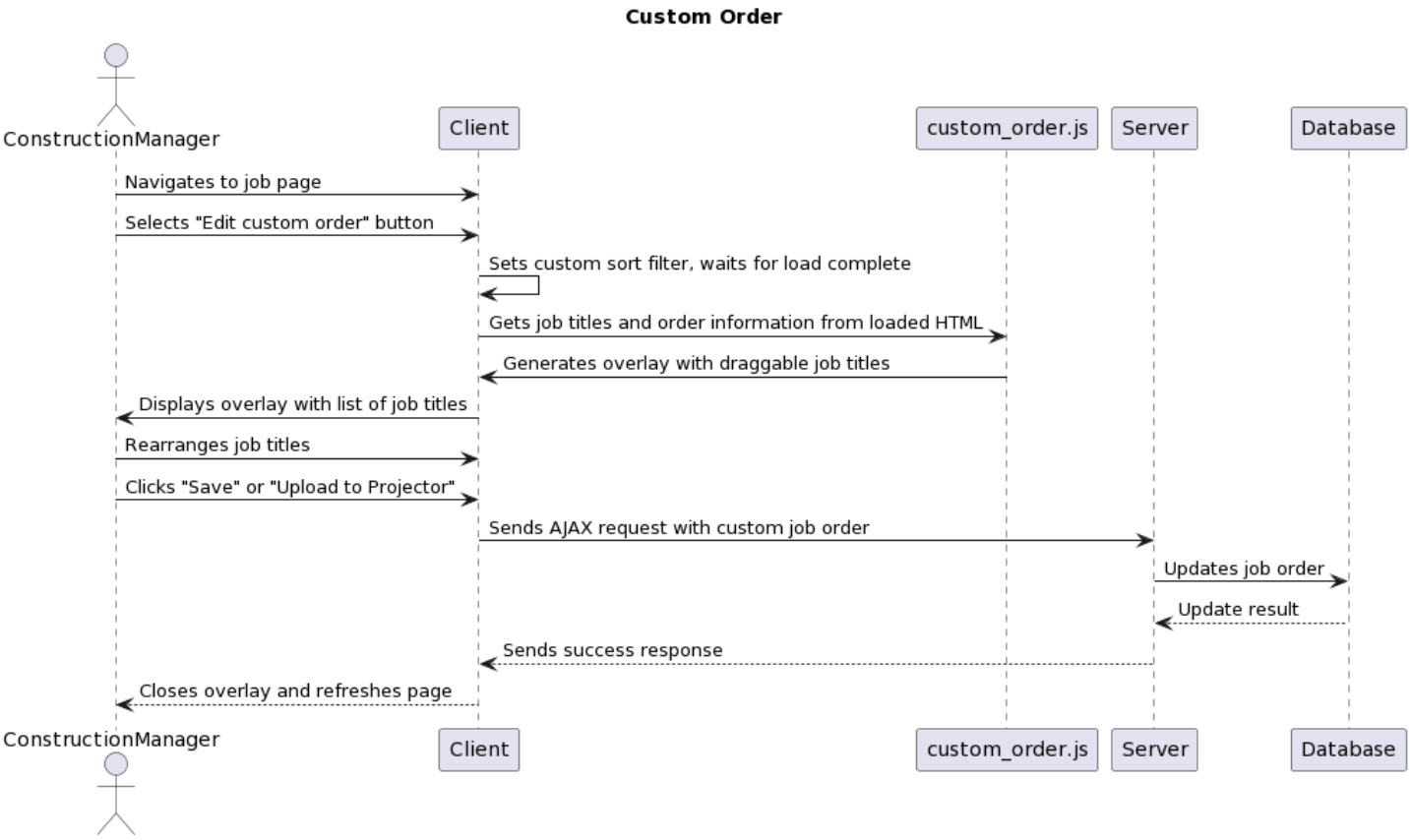
Page saves the order then closes the overlay and refreshes to display the new custom order.

Extensions:

7.1 User cancels

7.1.1 User decides not to save the custom order.

7.1.2 Overlay is closed without making changes.



The custom order function works by first setting the page to the custom sort option, this causes the page to retrieve the jobs and order of the jobs in the user's preferred custom sort. *custom_order.js* then pulls from the currently loaded page the order of the jobs as well as the job titles then generates an overlay with a list of the job titles that can be dragged and dropped to change the order of the jobs. When the user clicks save or clicks upload to projector, the page sends an ajax request to *save_custom_order.php* to save the specified order to either the user's personal account or the projector account. The page then closes the overlay and refreshes the page to display the new custom order.

37. View Projected vs Actual Employee Count Graph (Notable Feature)

Primary actor: Construction Manager

Description: Visual display of analytics related to jobs and previous employment numbers

Pre-condition: The user must be logged in to the system.

Post-condition: The user can view the analytics

Main scenario:

1. The user selects the history page of the software
2. The system populates a list of historical jobs with manpower summary/graph

38. View Where Last Assigned (Notable Feature) (**NEW**)

39. View History Page (**NEW**)

40. View Projected vs Actual Employee Count for Archived Jobs- History Page (Notable Feature) (**NEW**)

41. Show All Jobs- History Page (**NEW**)

42. Set/Disable Refresh Interval (**NEW**)

Primary actor: Construction Manager

Description: Allows the user to modify or disable their refresh interval (Use case 15)

Pre-condition: User must be logged in to the system and on their account page.

Post-condition: Their accounts refresh interval is updated in the database and in local storage

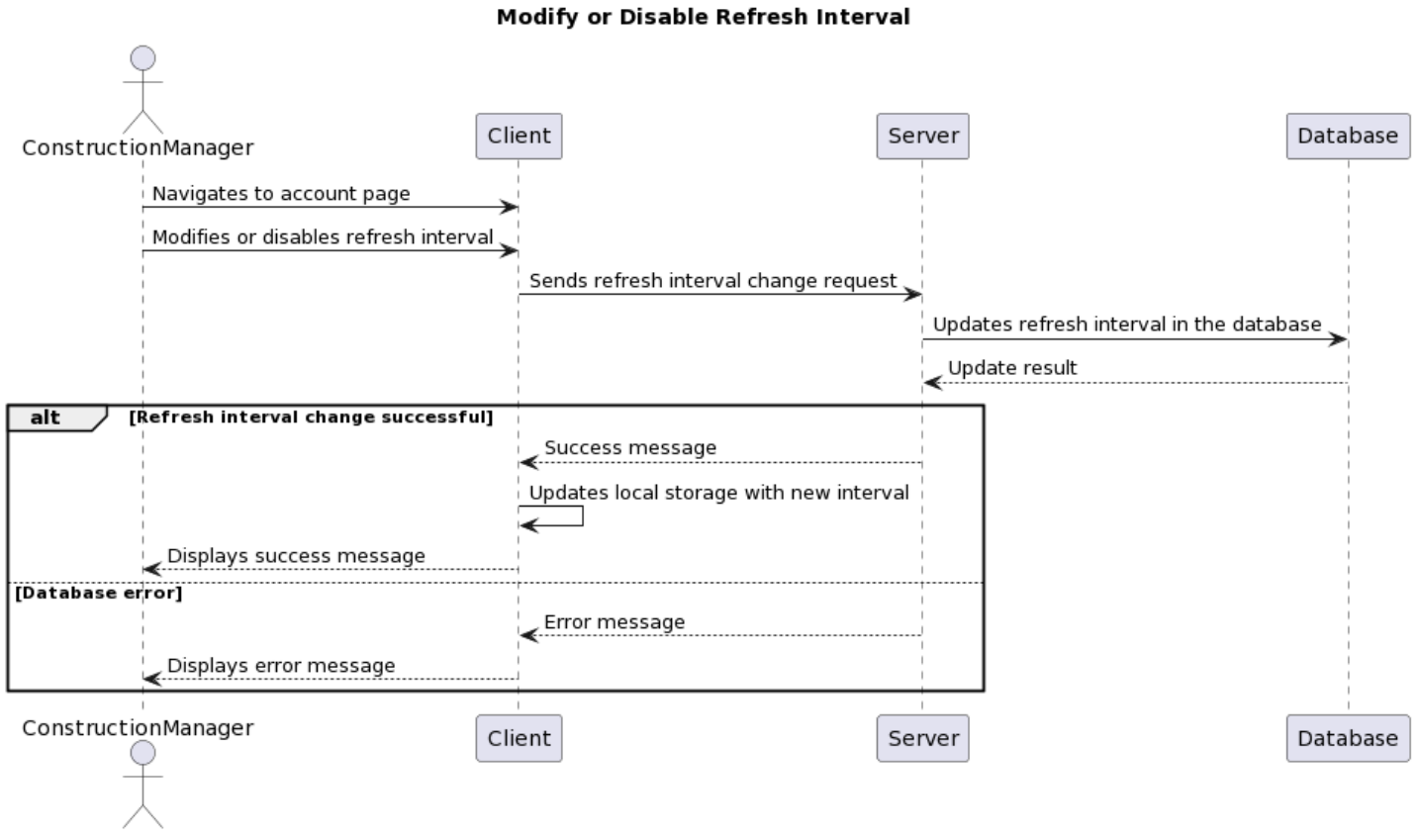
Main scenario:

1. User either changes or disables their refresh interval.
2. The selection is validated and then updated in the database.
3. The system notifies them of the success.

Extensions:

2.1. Database error

2.1.1. Database fails to update with new data due to connection issues or a failed database constraint.



The auto-refresh interval change function on the account page (*account.php*) sends a request to *set_refresh_interval.php* when the form is submitted. The *set_refresh_interval.php* then updates the database with the user's preferences. Upon successful completion, the account page updates the browser localstorage with the new interval.

43. View Help Page (**NEW**)

Primary actor: Construction Manager

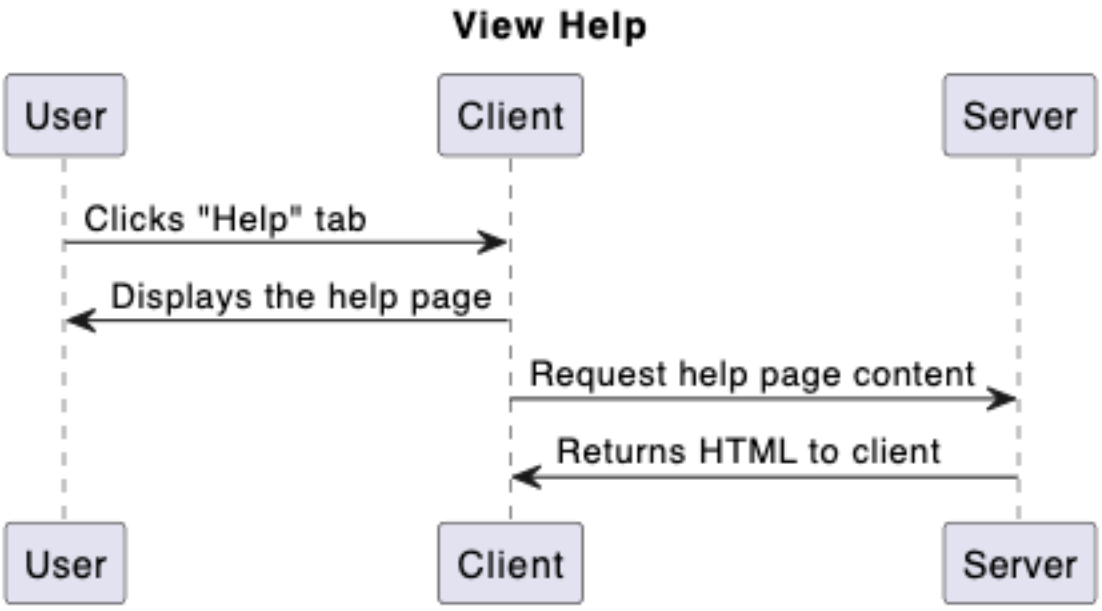
Description: Allows the user(user) to get help of how to use the system

Pre-condition: User must be logged in.

Post-condition: The help page is displayed to the user.

Main scenario:

1. User log in.
2. User clicks the "Help" tab
3. The help page is displayed to the user



The header of the page is generate by generate_nav_bar() function inside the global_generators.php. The page consists of htmls, the gif/png img elements are generated by generateFigureElement() function. The enlargered overlay window is generated by show_large_image() function. The gif-control.js allow the user click to start/stop the gif.

44. Pin/Unpin Employee[s] (**NEW**)

Primary actor: Construction Manager

Description: Allow the user to easily and quickly pin employees to the bottom pin bar

Pre-condition: The user must be logged in to the system and on the employee page

Post-condition: The employee is moved visually

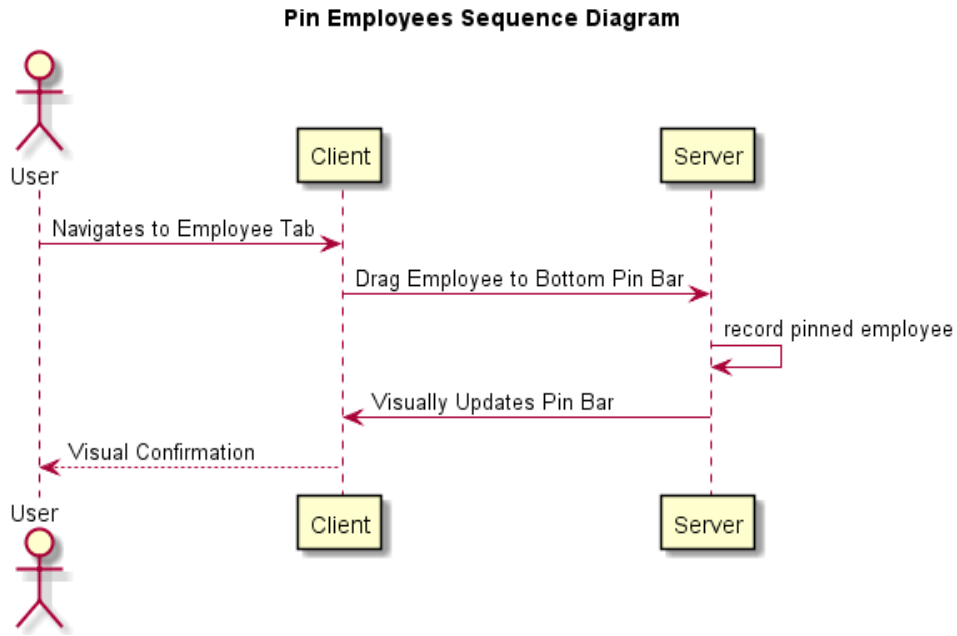
Main scenario:

1. User navigates to the employee tab
2. User drags employee to the bottom pin bar
3. System visually keeps the change.
4. System stages the pinned employees for future assignments

Extensions:

4.1. Database error

4.1.1. Database fails to move the employee due to connection issues or a failed database constraint.



45. Batch Add to Job (**NEW**)

46. Download Database Backup (**NEW**)

Primary actor: Construction Manager

Description: Allows the user to export a .sql file containing relevant job and/or employee data.

Pre-condition: User must be logged in to the system.

Post-condition: The .sql file is successfully exported by the system and downloaded to the client machine.

Main scenario:

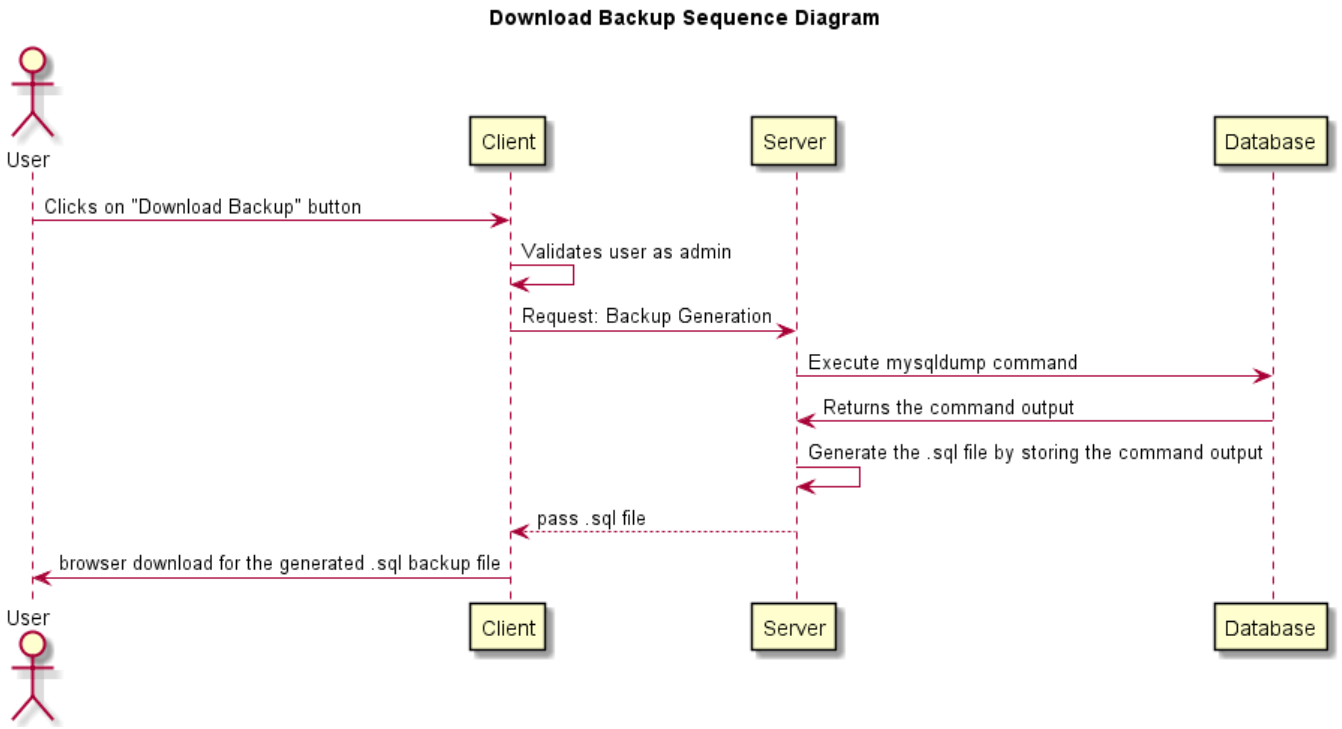
1. User selects the option to export a .sql file from the job or employee screen.

2. System processes database data and generates .sql file.
3. System sends generated file to client machine.
4. Client machine downloads file.

Extensions:

3.1. No data

3.1.1. There is no data to be exported on the current screen.



To download a backup file, the user must be logged in as an admin first, then that user can download a database backup by navigating to the functions page. Once the admin user clicks on the download backup file button, the client side calls *createBackupFile.php*, which executes a mysqldump command through web-server container's command line. The contents of that command are stored in a .sql file, which then gets passed to the front end as a browser download. The downloaded file should be located at the computer's downloads folder but might vary depending on the user's browser settings.

47. Upload Database Backup (**NEW**)

Use Case: Upload Database Backup

Primary Actor: Construction Manager

Description: Allows the user (Construction Manager) to upload a .sql file containing relevant job and/or employee data into the system for database restoration.

Pre-condition: User must be logged in to the system as a Construction Manager.

Post-condition: The .sql file is successfully uploaded and processed by the system, restoring relevant data to the database.

Main Scenario:

1. User selects the option to upload a .sql file for database restoration.
2. System prompts the user to choose a .sql file from their local machine.
3. User selects the desired .sql file.
4. System validates the user's role and file format (must be .sql).
5. System processes the uploaded .sql file and restores data to the database.
6. System displays a success message to the user indicating the successful data restoration.

Extensions:

6.1. Invalid File Format:

If the selected file is not in .sql format, the system notifies the user.

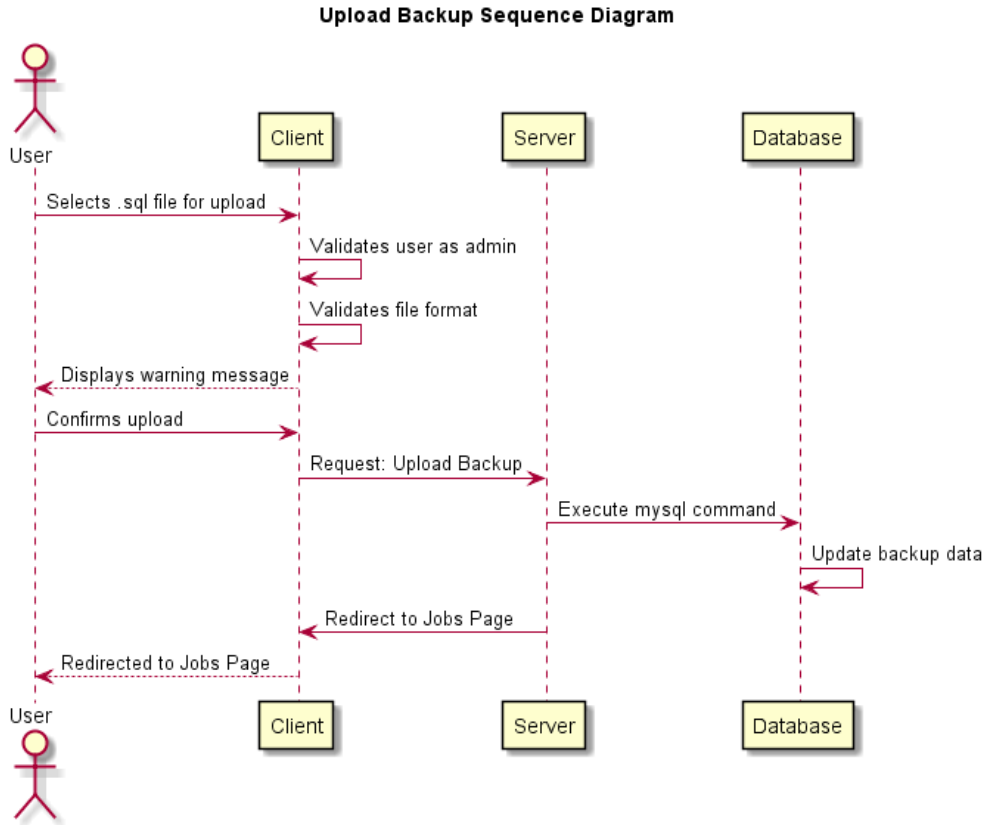
User is prompted to select a valid .sql file.

The scenario continues at step 3.

6.2. Data Restoration Failure:

If the data restoration process fails due to database inconsistencies or other issues, the system notifies the user.

User is prompted to review the .sql file for errors or contact system administrators for assistance.



To upload a backup file, the user must be logged in as an admin first, then that user can upload a database backup by navigating to the functions page. Once the admin user clicks on the select file button, the client side prompts a file upload window and lets the user choose a .sql file to upload. After a file is selected, the user then clicks on the upload button which triggers a client-side warning message. Upon user confirmation, the client side calls *loadBackup.php*, which executes a mysql command through web-server container's command line. After a successful load, the user is prompted a message and redirected to the jobs page.

48. Adjust Range- History Page (**NEW**)

49. Next/Previous Page + Change Jobs Per Page- History Page (**NEW**)

50. Change Username (**NEW**)

Primary Actor: Construction Manager

Description: The change username function allows the user to change their username within the system.

Pre-condition: User must be logged in to the system.

Post-condition: The user's username is successfully changed, and the system updates its records accordingly.

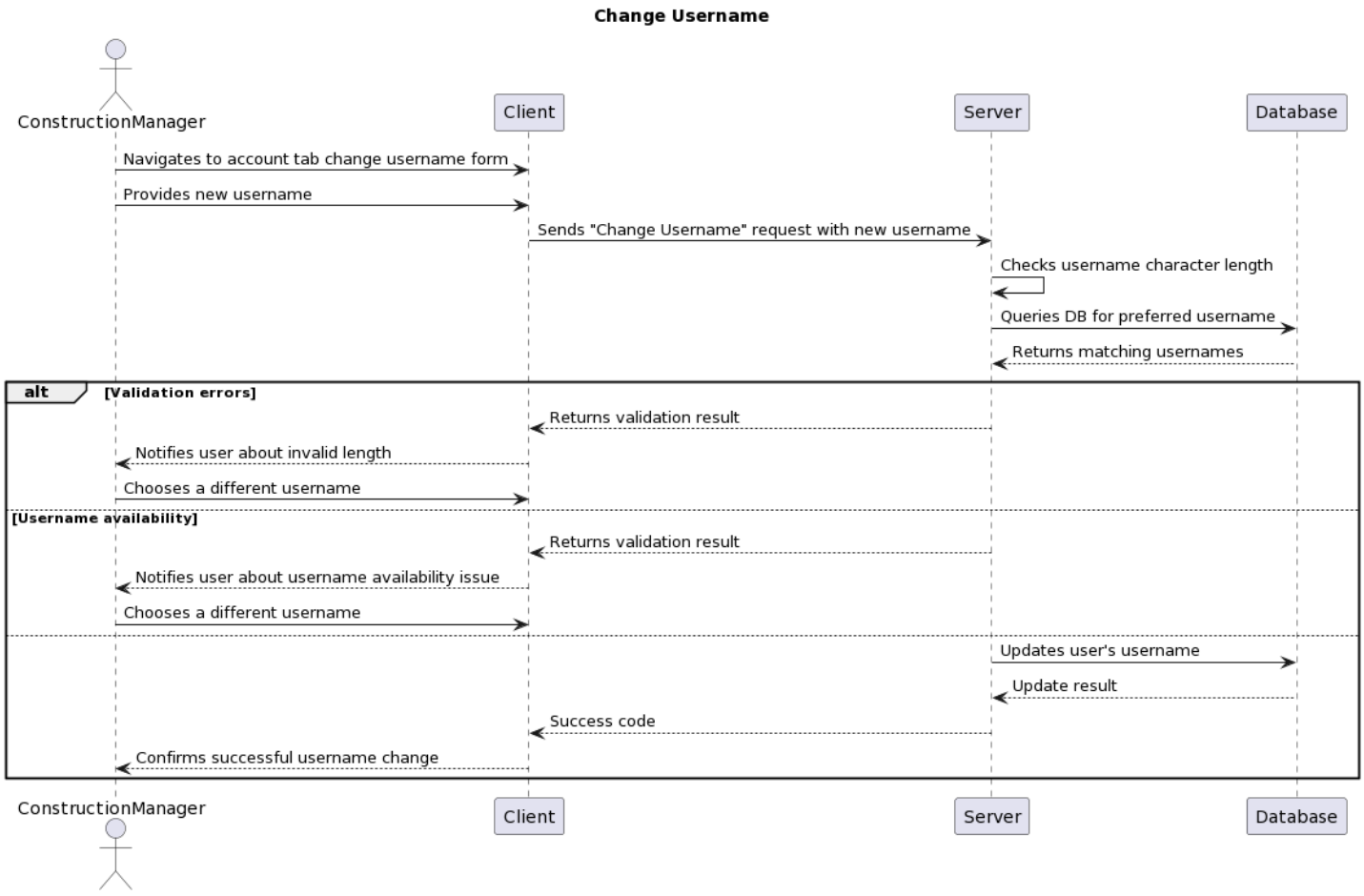
Main Scenario:

1. User decides to change their username.
2. User navigates to the "Change Username" option on the account tab.
3. User enters their new preferred username.
4. System verifies the new username availability and updates the user's username to the new one.
5. System confirms the successful username change to the user.

Extensions:

5.2 – Username is already taken/too long or too short

5.2.1 Message is displayed to the user indicating that username is taken or invalid length



51. Upload Job CSV (**NEW**)

Primary actor: Construction Manager

Description: Allows the user to upload a CSV file containing relevant data.

Pre-condition: User must be logged in to the system.

Post-condition: The CSV file is successfully uploaded and processed by the system.

Main scenario:

1. User selects the option to upload a CSV file.
2. User chooses the desired CSV file from their local machine.
3. User confirms the file selection.
4. System validates the file format and content.
5. System processes the CSV file and updates the relevant data in the system.

Extensions:

2.1. Invalid file format

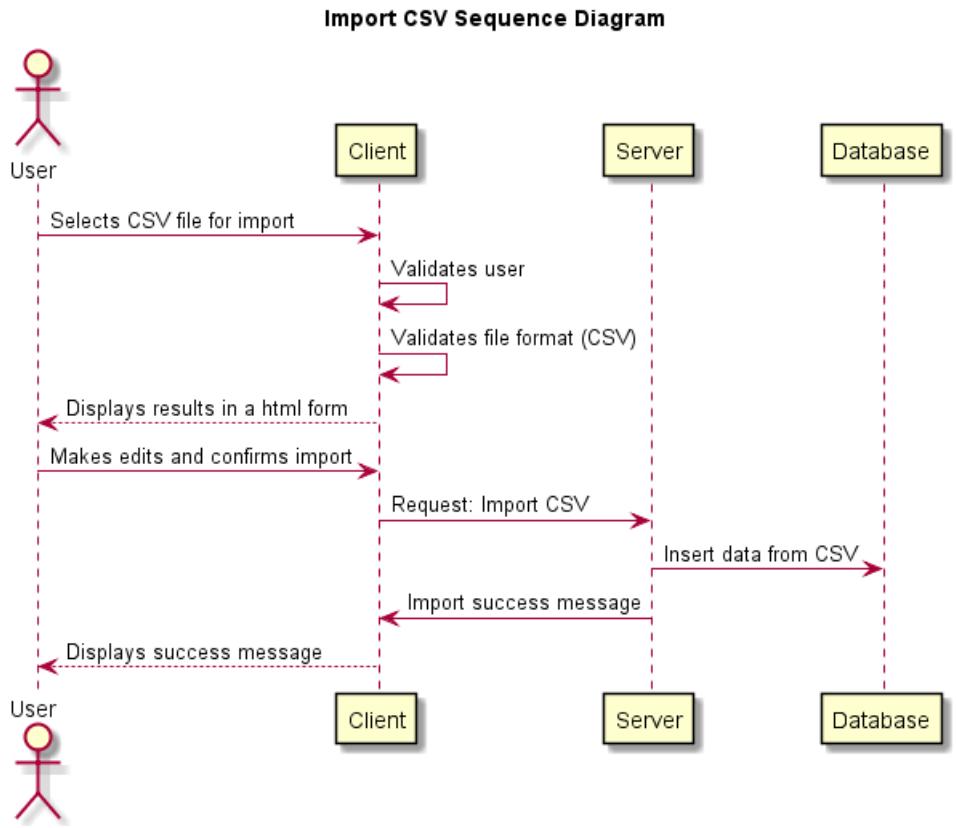
2.1.1. The user selects a file that is not in CSV format. The system notifies the user about the invalid file format.

2.2. Empty file

2.2.1. The user selects a CSV file that is empty. The system notifies the user that the file is empty.

2.3. Data validation failure

2.3.1. The system identifies errors or inconsistencies in the CSV file data. The system provides error messages or prompts the user to correct the data before proceeding.



The Client validates the CSV format and the user login status, then reads the contents of the csv file and displays them in a form by calling *processUploadJobsCSV.php*. After User confirmation, the Client requests CSV import from the Server by passing the data to *insertJobCSV.php*. The Server inserts CSV data into the Database by executing a mysql command, then signaling success to the Client by displaying a confirming message to the User.

Upload Employee CSV

Primary actor: Construction Manager

Description: Allows the user to upload a CSV file containing relevant data.

Pre-condition: User must be logged in to the system.

Post-condition: The CSV file is successfully uploaded and processed by the system.

Main scenario:

6. User selects the option to upload a CSV file.
7. User chooses the desired CSV file from their local machine.
8. User confirms the file selection.
9. System validates the file format and content.
10. System processes the CSV file and updates the relevant data in the system.

Extensions:

2.1. Invalid file format

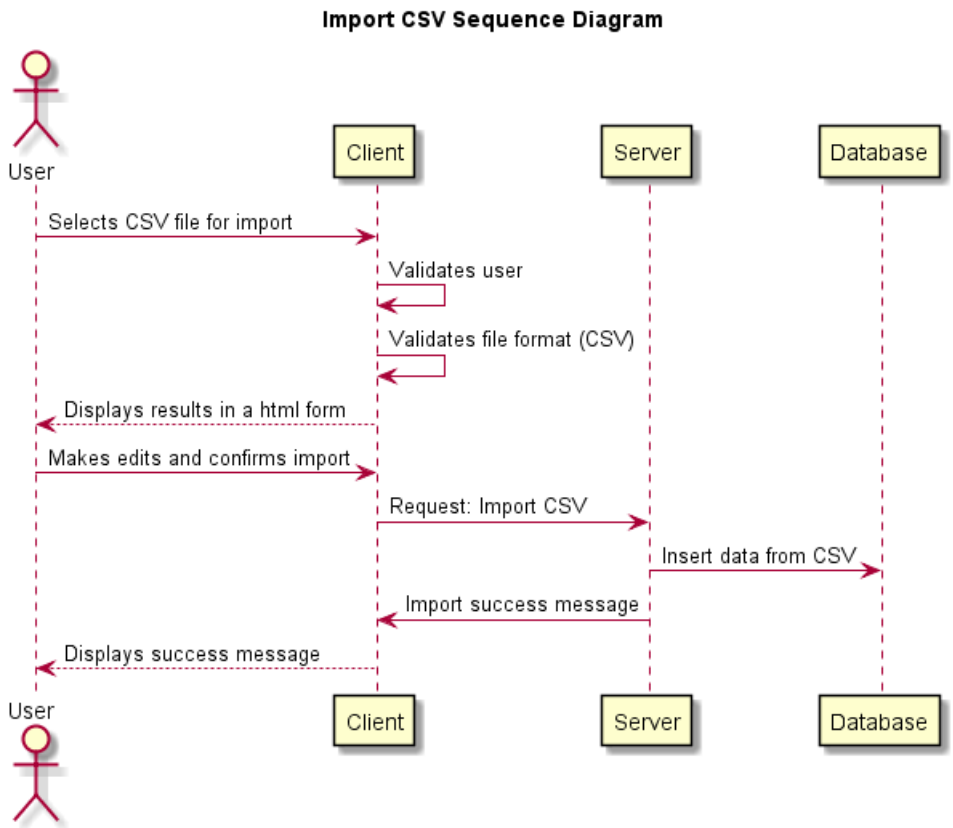
2.1.1. The user selects a file that is not in CSV format. The system notifies the user about the invalid file format.

2.2. Empty file

2.2.1. The user selects a CSV file that is empty. The system notifies the user that the file is empty.

2.3. Data validation failure

2.3.1. The system identifies errors or inconsistencies in the CSV file data. The system provides error messages or prompts the user to correct the data before proceeding.



The Client validates the CSV format and the user login status, then reads the contents of the csv file and displays them in a form by calling *processUploadEmployeeCSV.php*. After User confirmation, the Client requests CSV import from the Server by passing the data to *insertEmployeeCSV.php*. The Server inserts CSV data into the Database by executing a mysql command, then signaling success to the Client by displaying a confirming message to the User.

52. View Documentation (**NEW**)

Primary actor: Admin

Description: Allows the admin user to view the documentation for the system

Pre-condition: User must be logged in to the system as an admin

Post-condition: The documentation is displayed to the admin user.

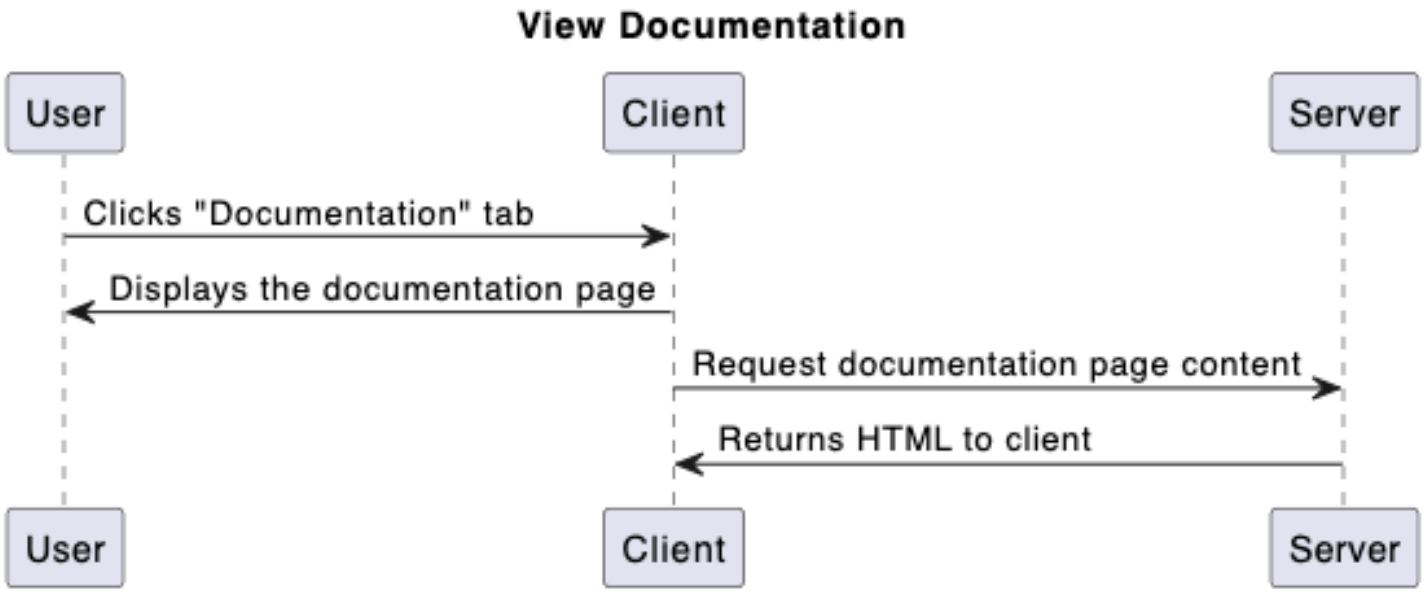
Main scenario:

4. User log in.
5. User clicks the “Documentation” tab
6. The documentation is displayed to the user

Extensions:

1.1. The user is not an admin user

2.1.1. The window displays a “You do not have permission to access this page or perform this action” message



When a user clicks the documentation tab, documentation.php calls the check_is_admin() function inside the session_security.php, which ensures only the admin user can access the page. The header of the page is generate by generate_nav_bar() function inside the global_generators.php. The page consists of htmls, the gif/png img elements are generated by generateFigureElement() function. The enlargered overlay window is generated by show_large_image() function. The gif-control.js allow the user click to start/stop the gif.

53. Change Background Image (**NEW**)

Primary actor: Construction Manager

Description: Allows the user to change their background image

Pre-condition: User must be logged in to the system.

Post-condition: The background image is changed per user

Main scenario 1:

1. User select/drag image to be uploaded.
2. System validates the format and the size of the image
3. User clicks the “upload selected” button.

4. Background changes for user

Extensions:

2.1. Invalid file format

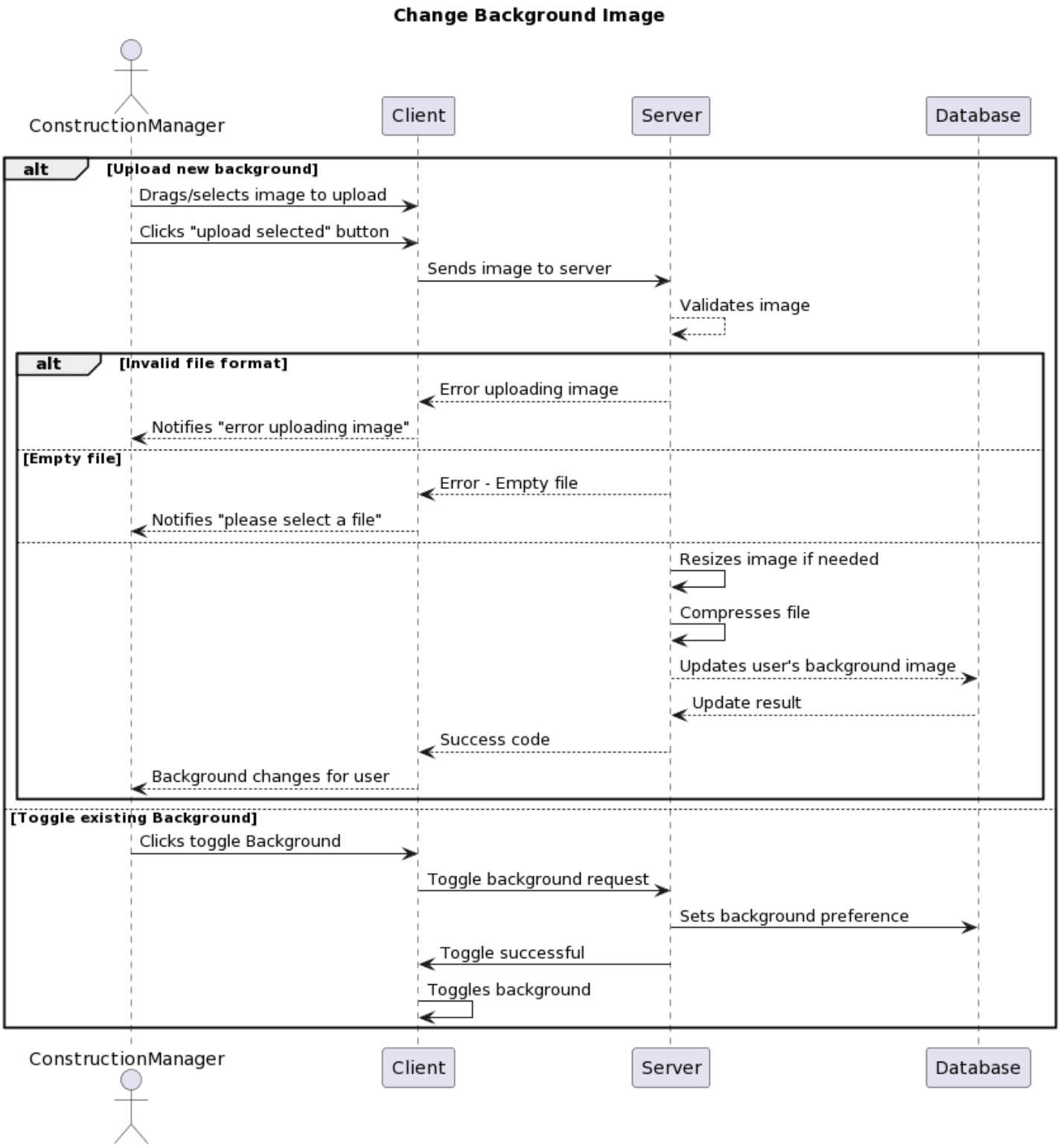
2.1.1. The user selects a file that is not in png/gif/jpeg/jpg format, the system will notify the user with a “error uploading image” error message.

2.2. Empty file

2.2.1. if no image dragged/selected, the system notifies with a “please select a file” message.

2.3. File too big

2.3.1 If the file is bigger than 1440p width, the system will resize the image to be max 1440p width.



The change background function is a form on *account.php*. When the form is submitted, the information is sent via an ajax request to *change_background.php* who then validates the image is an image and compresses the file using the imagik library. The file is saved to the server as *user_id.png*. After a successful upload of the file, the server returns a success message and the client enables the background for the user.

Removed Use Cases (out of scope)

54. Undo (**REMOVED**)

55. Export CSV (**REMOVED**)

56. View Diagnostic Data (**REMOVED**)

Project Management

Features List

Table 1. Feature List

	Features	Status	Note
1	View Job Page	completed	
2	Allocate/Move Employee	completed	
3	Add New Job	completed	
4	Add New Employee	completed	
5	Edit Outlook (Projection)	completed	
6	View Employee List	completed	not really use case
7	Show All/Unassigned Employees - Job page - via employee list	completed	
8	Change Active Status [Active/Inactive/School]	completed	
9	View Employee Details	completed	
10	Edit Employee Details	completed	
11	View Job Details	completed	
12	Edit Job Details	completed	
13	Delete Employee	completed	
14	Delete Job	completed	
15	View Employee Totals	completed	not really use case
16	Auto Refresh	completed	not really use case
17	Duplicate Employee	completed	
18	Login/Admin Login	completed	
19	Logout	completed	
20	View Manager/Admin Navbar	completed	not really use case
21	View Account Page	completed	
22	Change Password	completed	
23	View Presenter View	completed	
24	View Admin Functions	completed	
25	Create Manager Account	completed	
26	View Employee Page	completed	
27	Archive Job	completed	
28	Unarchive Job	completed	

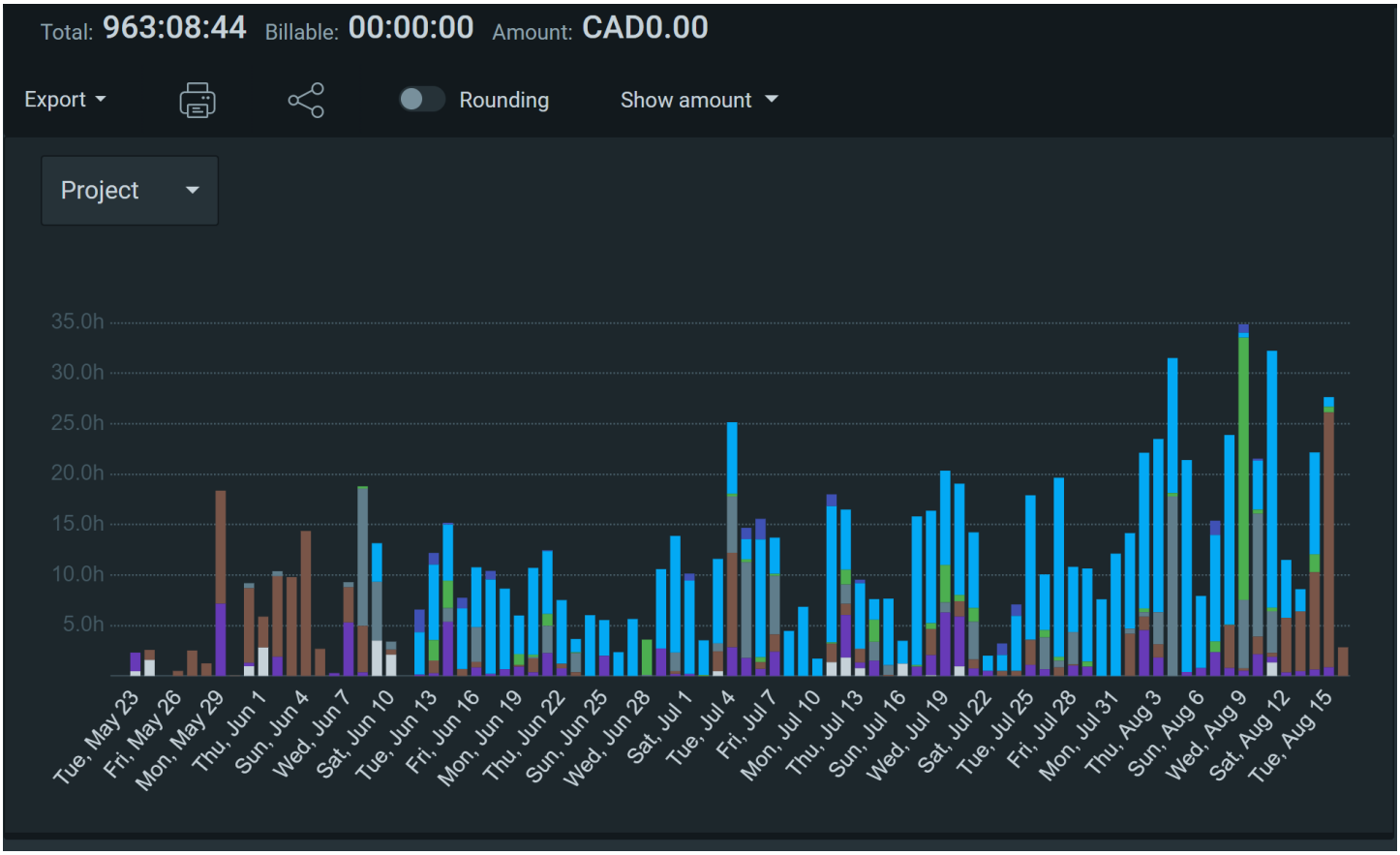
29	Show Archived Employees	completed	
30	Archive Employee	completed	
31	Unarchive Employee	completed	
32	Load Database Backup	completed	
33	Reset User's Password	completed	
34	Sort/Filter/Search Employees	completed	
35	Sort/Filter/Search Jobs	completed	
36	Upload Employee Image	completed	
37	Custom Sort Jobs [+ upload to projector]	completed	
38	View Projected vs Actual Employee Count Graph	completed	not really use case
39	View Where Last Assigned	completed	not really use case
40	View History Page	completed	
41	View Projected vs Actual Employee Count for Archived Jobs - History Page	completed	not really use case
42	Show All Jobs - History Page	completed	
43	Set/Disable Refresh Interval	completed	
44	View Help Page	completed	
45	Pin Employee[s]	completed	
46	Batch Add to Job	completed	
47	Unpin Employee[s]	completed	
48	Download Database Backup	completed	
49	Upload Database Backup	completed	
50	Adjust Range - History Page	completed	
51	Next/Previous Page + Change Jobs Per Page - History Page	completed	
52	Change Username	completed	
53	Upload Job CSV	completed	
54	Upload Employee CSV	completed	
55	View Documentation	completed	
56	Change Background Image	completed	
57	Undo	removed	Too much opportunity cost
58	Export CSV	removed	Not needed
59	View Diagnostic Data	removed	Not really needed

Actual VS Expected Hours for Each Member

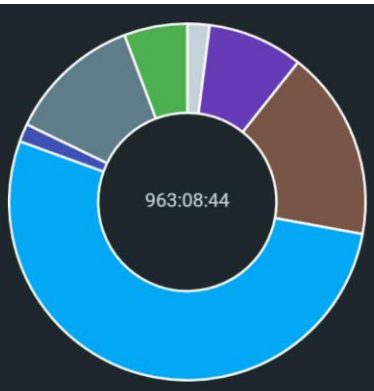
Table 2

Week #	Start Date	End Date	Yuan(hours)	Adam(hours)	Eddy(hours)	Edwin(hours)	Jordan(hours)	Total	Expected Total	Note
1	2023-05-21	2023-05-27	3.90	1.52	0.00	2.50	0.00	**7.92**		
2	2023-05-28	2023-06-03	8.72	19.02	10.25	14.50	2.43	**54.92**	90.00	
3	2023-06-04	2023-06-10	11.52	16.60	14.22	14.50	5.17	**62.00**	90.00	
4	2023-06-11	2023-06-17	8.97	13.82	16.83	13.03	10.20	**62.85**	90.00	
5	2023-06-18	2023-06-24	6.38	6.25	15.75	13.65	12.98	**55.02**	90.00	
6	2023-06-25	2023-07-01	2.25	13.83	11.73	9.80	14.12	**51.73**	90.00	
7	2023-07-02	2023-07-08	11.52	34.82	13.22	18.32	10.80	**88.67**	90.00	
8	2023-07-09	2023-07-15	7.57	23.00	12.18	15.13	9.98	**67.87**	90.00	
9	2023-07-16	2023-07-22	5.67	34.17	14.55	22.17	14.82	**91.37**	90.00	
10	2023-07-23	2023-07-29	11.93	23.42	14.78	20.18	9.05	**79.37**	90.00	
11	2023-07-30	2023-08-05	18.23	39.43	26.07	27.65	21.02	**132.40**	90.00	
12	2023-08-06	2023-08-12	28.15	44.52	23.38	30.13	21.20	**147.38**	90.00	
13	2023-08-13	2023-08-16	11.50	24.50	16.00	18.50	9.25	**79.75**	45	not full week, estimate
Total			**136.30**	**294.88**	**188.97**	**220.07**	**141.02**	**981.23**	1,035.00	

Clockify Stats

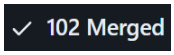


	TITLE	DURATION	AMOUNT
11	(Without project)	19:25:47	CAD0.00
108	Communication	83:24:59	CAD0.00
65	Documentation	165:44:15	CAD0.00
247	Implementation	507:02:25	CAD0.00
16	Misc	15:45:27	CAD0.00
63	Peer Evals/Presentations/Scott	116:35:32	CAD0.00
49	Process-Related	55:10:19	CAD0.00

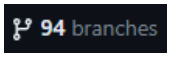


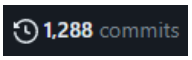
TEAM MEMBER	LATEST ACTIVITY			TOTAL TRACKED (MAY 23 - AUG 16)	
AD Adam	Final Documentation Documentation	00:02:13	In progress	289:16:58	
EZ Eddy Zhang	final doc Documentation	02:18:06	In progress	185:56:52	
YU Yuan	final documentation Documentation	00:00:01	30 minutes ago	134:33:13	
ED Edwin	Working on docs Documentation	00:46:22	3 hours ago	214:18:49	
JO Jordan	documentation Documentation	03:09:14	7 hours ago	139:02:52	

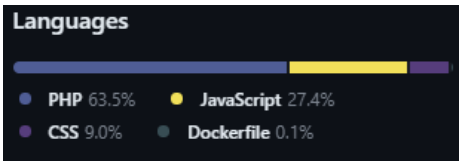
Github Stats

- 102 pull requests merged 

- 228 tracked tasks completed 

- 94 branches 

- >1280 commits 

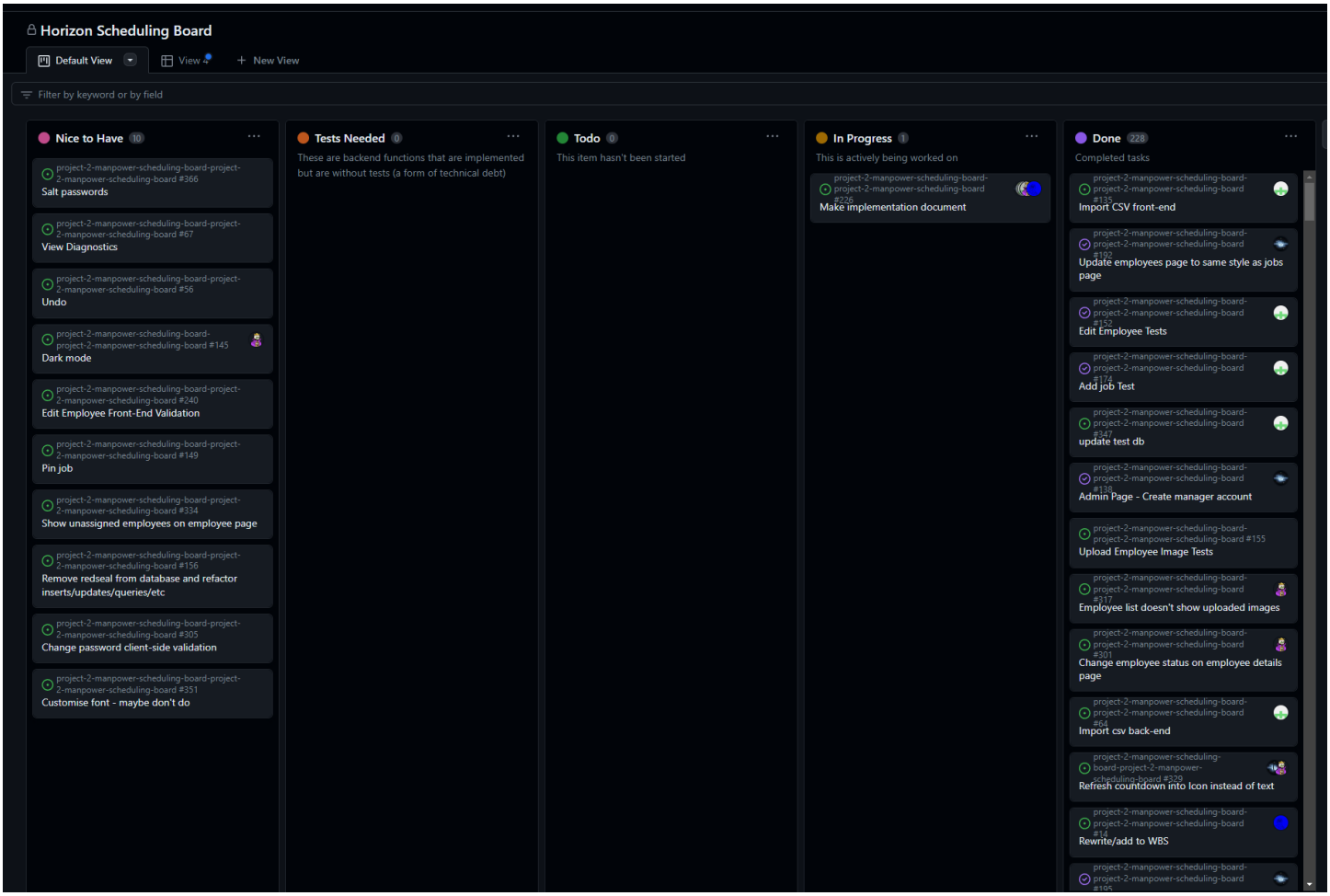
- 

Total Builds

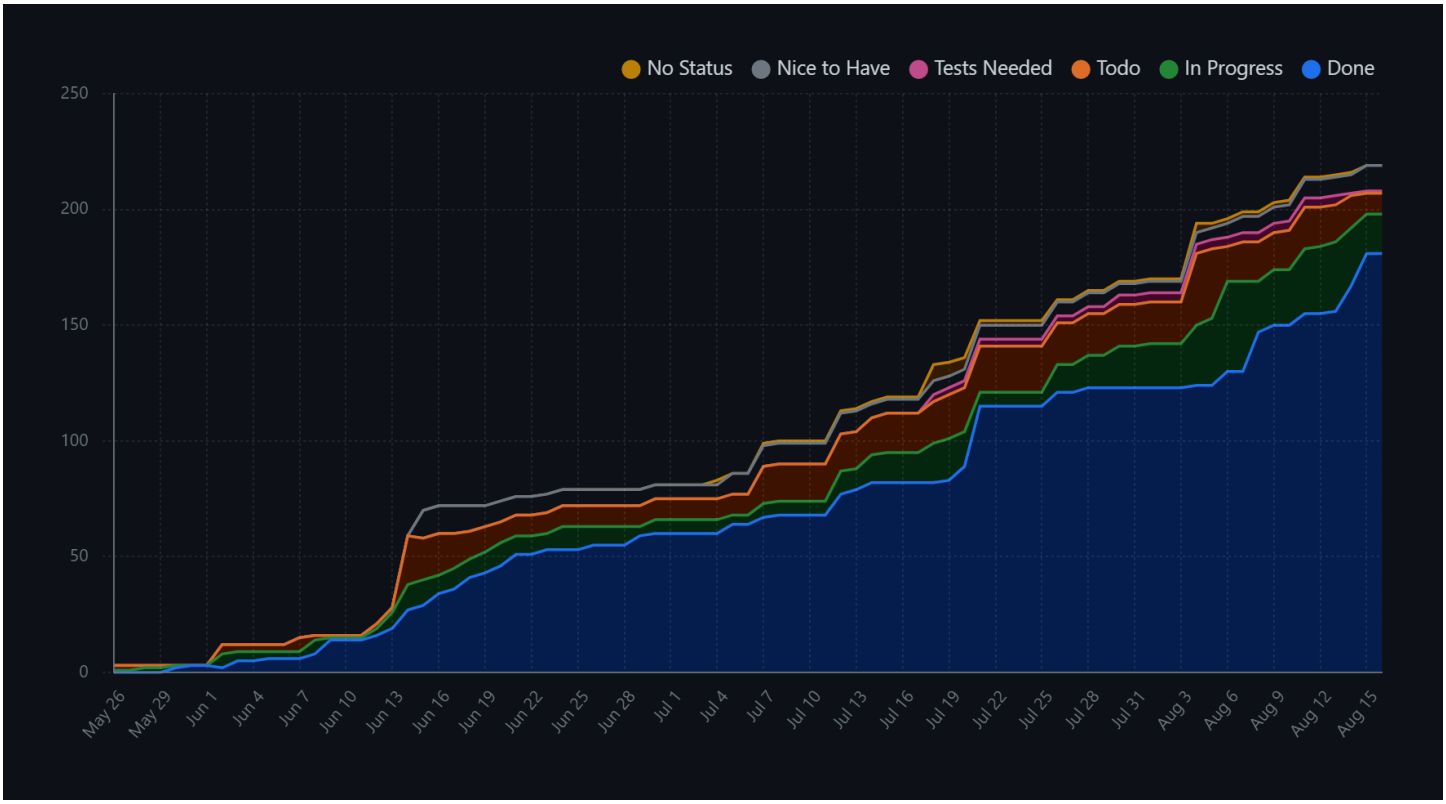
- Drone Builds: 509

- Lines of code (warning: really not all that accurate compared to effort put in, but still interesting):

Kanban Board



Burnup Chart



Note: Github project burnup charts seem to always be a little bit inaccurate, but the rough shape with the done is accurate.

Status of the Software Implementation

Testing

Requirement	Type of test Unit test(UNT), Integration test(IT), User test(UT)	Pass/Fail	Contributer
Add New Job			
The system presents a form to prompts the user to enter the name, duration (start month, end month), and the project manager (optional field) of the new job/project.	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system validates the entered information to ensure it meets any defined constraints.	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system stores the job/project details in the database.	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Add New Employee			
The system presents a form to prompts the user to enter the details of the new employee.	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system validates the entered information to ensure it meets any defined constraints.	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system stores the employee details in the database.	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Allocate Employee			
User drags employee from job to other job	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
System visually keeps the change.	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
System updates the database with the change	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Edit Number of Employees for the job month.			
User manually type the number of employees for a job month.	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system validates inputs and updates the number of employees allocated for the job month.	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan

Log in			
System validates user credentials	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
System logs user in by Session	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Log out			
System destorys the session and redirects user to login page	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
View Editable schedule			
The system displays the schedule on the user interface, presenting tasks, employees, start and end dates, and other relevant details.	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system takes user input and updates the project database with the modified schedule information	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
View Simplified Schedule (projector view)			
The system displays the schedule on the user interface, presenting tasks, employees, start and end dates, and other relevant details.	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Edit Job Detail			
The system displays the current details of the selected job/project, such as name, start and end date, and other relevant information.	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system validates the updated information to ensure it meets any defined constraints.	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system updates the job's/project's details in the system.	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
View Graph/Summaries			
The system populates graphs/sumaries depending on user selection	UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Archive Job			
The system unassign all current employess working on that job	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system moves the job to the archived table	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan

Delete Employee			
The system validates that the employee is not assigned to a job	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system deletes the employee record from the database	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Import jobs CSV file			
The system takes user uploaded csv file and process it's data	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system displays a HTML form of the data to allow user to edit them	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system takes submission of the displayed html table and insert jobs in the database	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Download Backup			
The system processes database data and generates CSV file and save the file to client machine	UNT, IT	Fail(dependency)	Eddy, Adam, Edwin, Jordan, Yuan
Create Manager Account			
The system validate the logged in user is an admin	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system validates user input and create a new user in the database	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Change Password			
System verifies that the current password is correct and updates the user’s password to the new one.	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Reset User Password			
The system validate the logged in user is an admin	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system resets the selected user’s password to a default value	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
View Employee List			
The user clicks on the 'employee list' button	UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan

The system opens the list and populates the results	UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Show All/Unassigned Employees- Job page- via employee list			
The user clicks on the 'Employee List' button.	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system opens the list and populates the results	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The user may then Move/Allocate an employee (Use Case 2) or View Employee Details (Use Case 8)	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Change Active Status [Active/Inactive/School]			
The user drags an employee's card into the respective area	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system updates that employee's active status	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
View/Edit Employee Details			
The system grabs all relevant Employee information from the database and display to the user	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The User may edit the saved employee details or close the overlay, relevant changes will be reflected in the database	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
View/Edit Job Details			
The system grabs all relevant information from the database and display to the user	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The User may edit the saved Job details or close the overlay, relevant changes will be reflected in the database	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Delete Job			
The system prompts the user a warning	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system unassign employees from that job and deletes the job record from the database after user confirmation	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
View Employee Totals			
The system displays both the current number of and the projected number of employees on a job	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Auto Refresh			

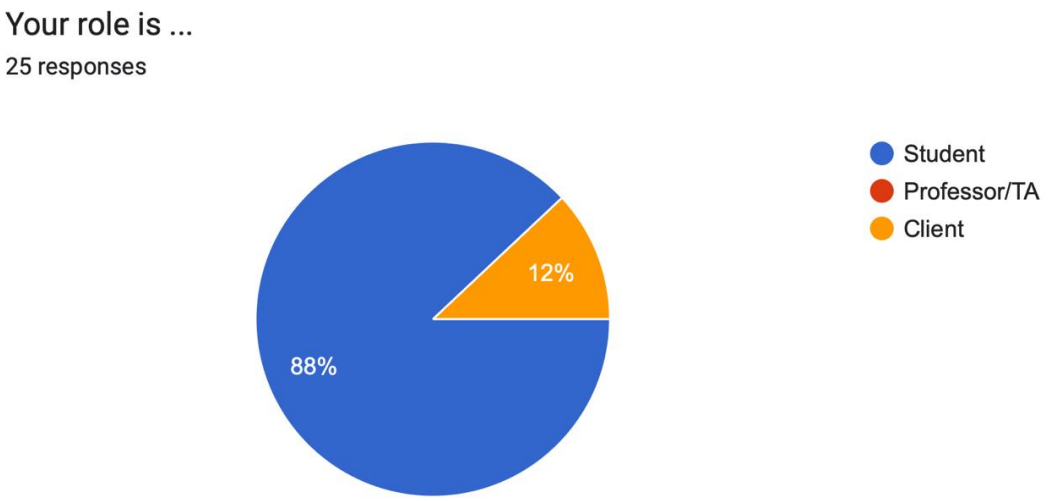
The interval time finishes and checks for new updates, then repopulates the page with new changes	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Duplicate Employee			
The system duplicates the employee card upon user request	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system updates the projected employee number	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
View Account Page			
The system populates the account page with relevant forms	UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
View Presenter View			
The system displays the schedule on the user interface, presenting tasks, employees, start and end dates, and other relevant details.	UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
View Admin Functions			
The system validates that the user is an admin	UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system displays the admin-only functions	UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
View Employee Page			
The system navigates to the employee page and populate it with relevant data upon user request	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Unarchive Job			
The System prompts the user if they are sure	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system changes the is_archived field of an archived job to null	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Show Archived Employees			
the system displayes all of the archived employees upon user request	UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Archive Employee			

The System prompts the user if they are sure	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system changes the is_archived field of an employee to the archive date	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Load databse backup			
The system validates the file upload	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system ecexutes a mysql command to load the backup	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Reset User's Password			
The system validates that the user is an admin	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system resets the selected user's password to a temporary value and prompts a success message	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Sort/Filter/Search Employees			
The system repopulates the employees page depending on the sort/filter/search settings	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Sort/Filter/Search Jobs			
The system repopulates the job page depending on the sort/filter/search settings	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Upload Employee Image			
The system validates the file upload	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system compresses and saves the uploaded image	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system displayes the newly uploaded user image for that employee	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Custom Sort Jobs			
The system repopulates the job page based on the order the the user decides	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
View History Page			
The system populates the page based on archived jobs with relevant graphs	UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan

The system successfully repopulates the page and graphs based on user settings	UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Set/Disable Refresh Interval			
The system sets/disables refresh interval based on user input	UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
the system prompts a success messgae after user input is set	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Pin Employee			
The system pins employee cards to the bottom of the page based on user input	UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Batch Add to Job			
The system assigns all pinned employee to the designated job	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Unpin Employee			
The system unpins employee cards from the bottom of the page based on user input	UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Download Database Backup			
The system validates that the user is an admin	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system processes database data and generates .sql file.	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system triggers a browser download with the generated data	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Upload Database Backup			
The system validates that the user is an admin	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system validates the file upload	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system prompts a warning message to the user before loading the backup file	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system reads the data inthe file and executes a mysql command to load the contents of the file	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system prompts a success/failure message and redirectes the user to the job page	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan

Import Employee CSV file			
The system takes user uploaded csv file and process it's data	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system displays a HTML form of the data to allow user to edit them	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system takes submission of the displayed html table and insert the employees in the database	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
View Documentation			
The system displays documentation and videos on the documentation page	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
Change Background Image			
The system validates the uploaded image file	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
The system compresses the uploaded image and saves the compressed image	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan
the system displays the compressed image as the background	UNT, IT, UT	Pass	Eddy, Adam, Edwin, Jordan, Yuan

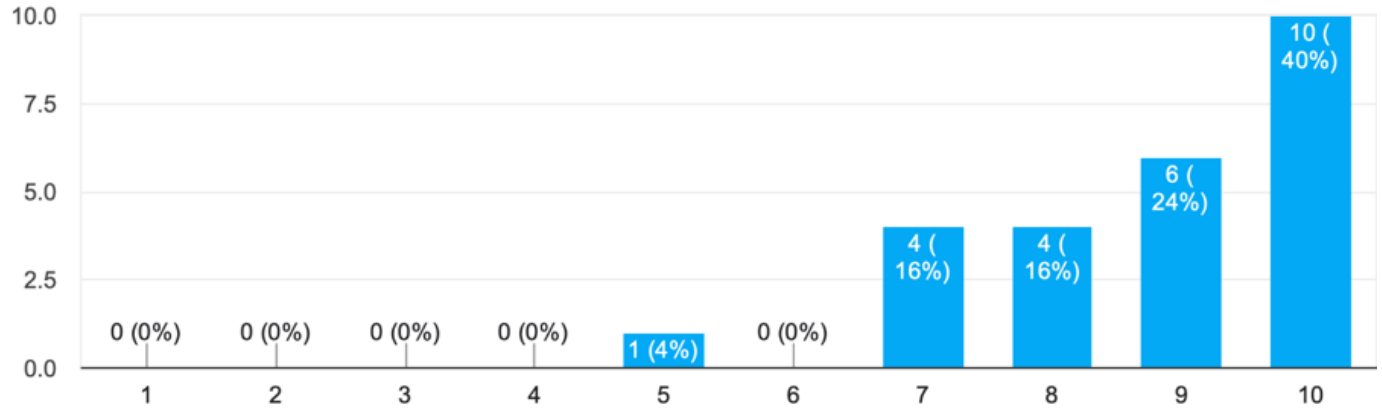
User Testing



SUS Questions:

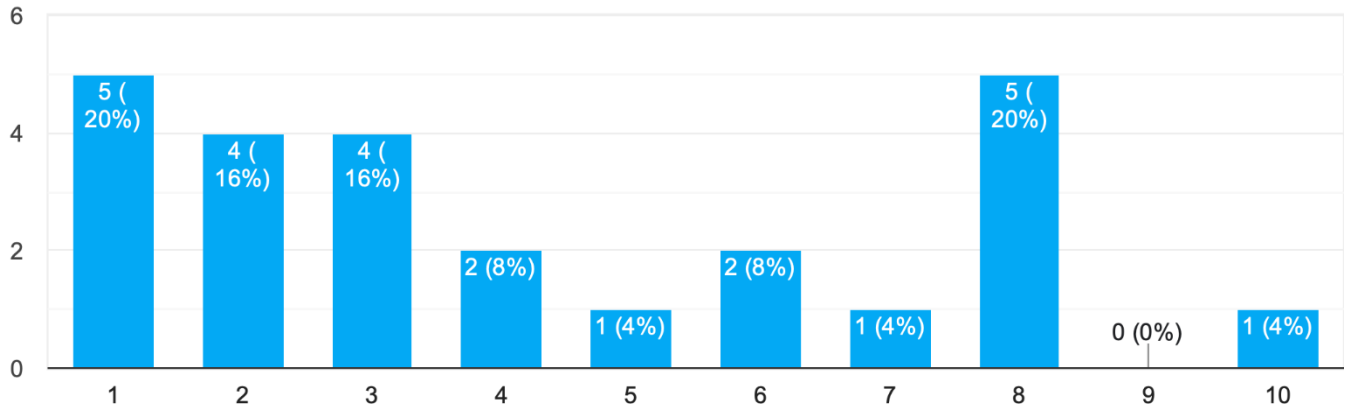
I think that I would like to use this system frequently.

25 responses



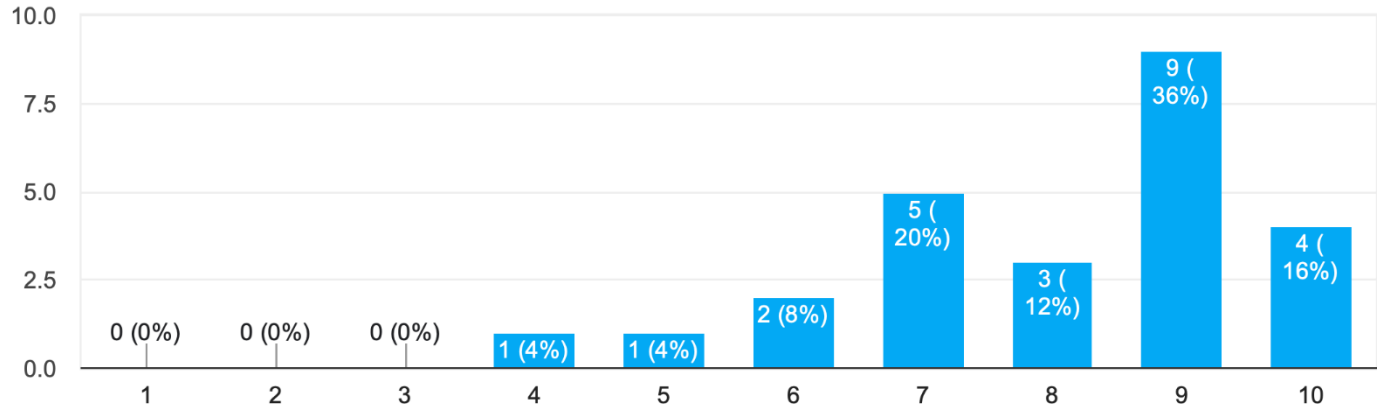
I found the system unnecessarily complex.

25 responses



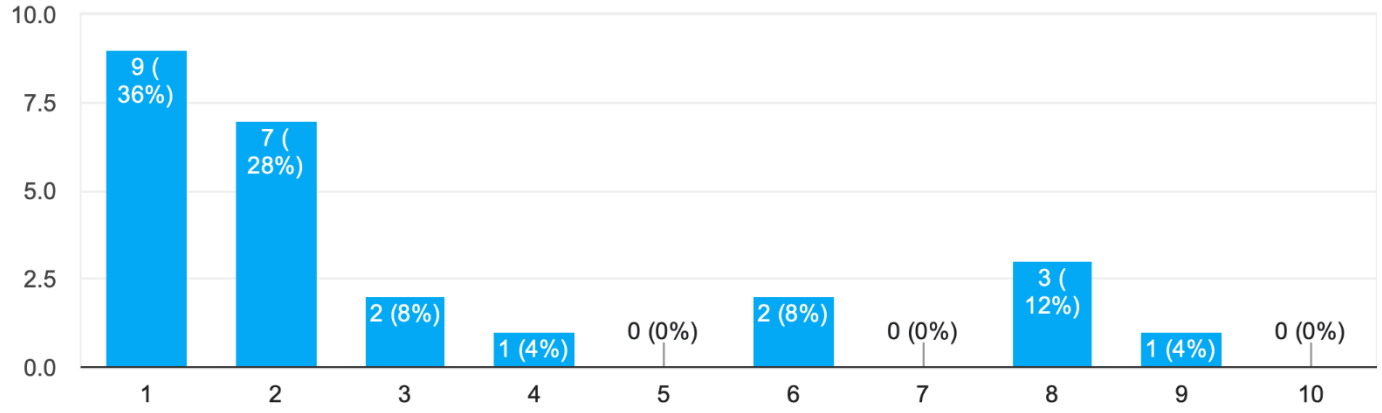
I thought the system was easy to use.

25 responses



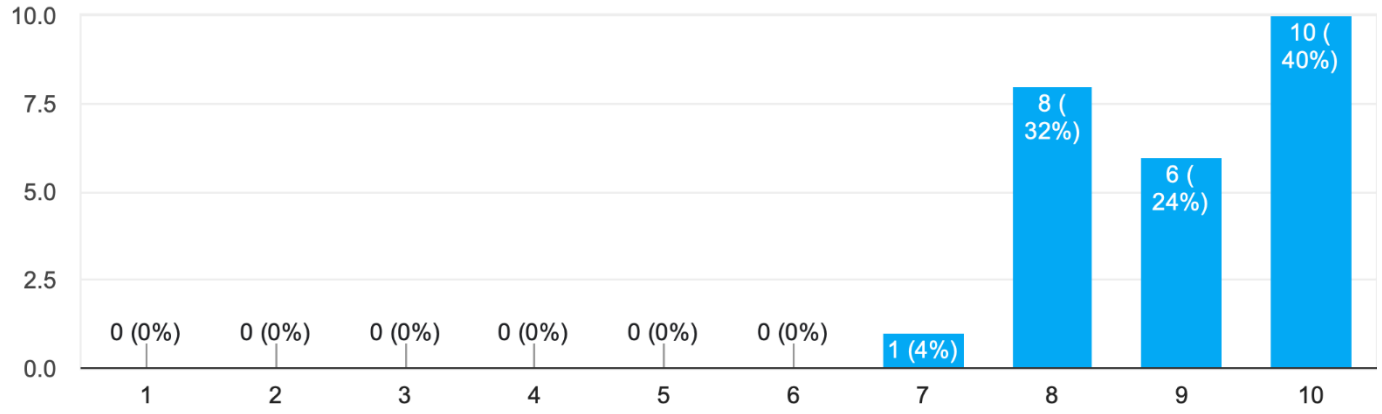
I think that I would need the support of a technical person to be able to use this system.

25 responses



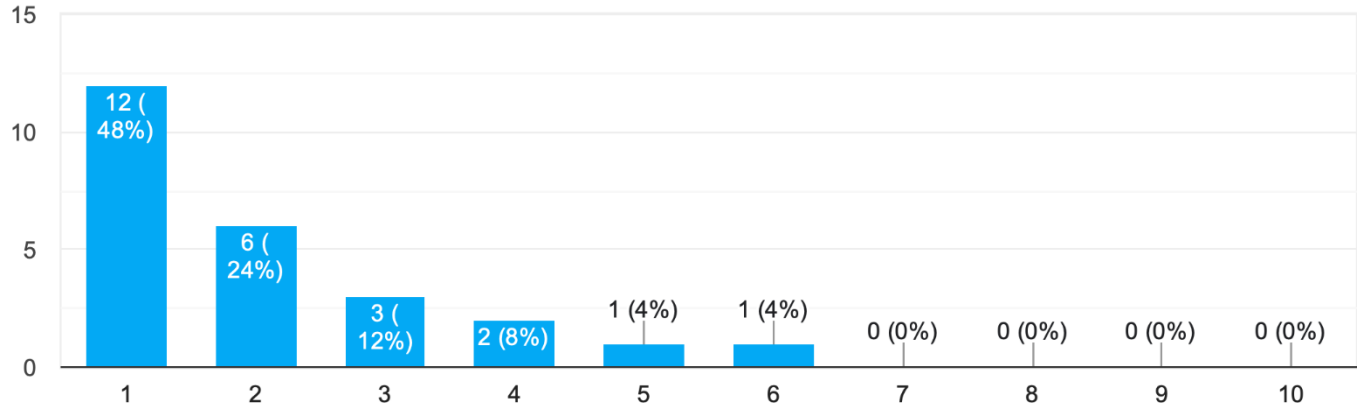
I found the various functions in this system were well integrated.

25 responses



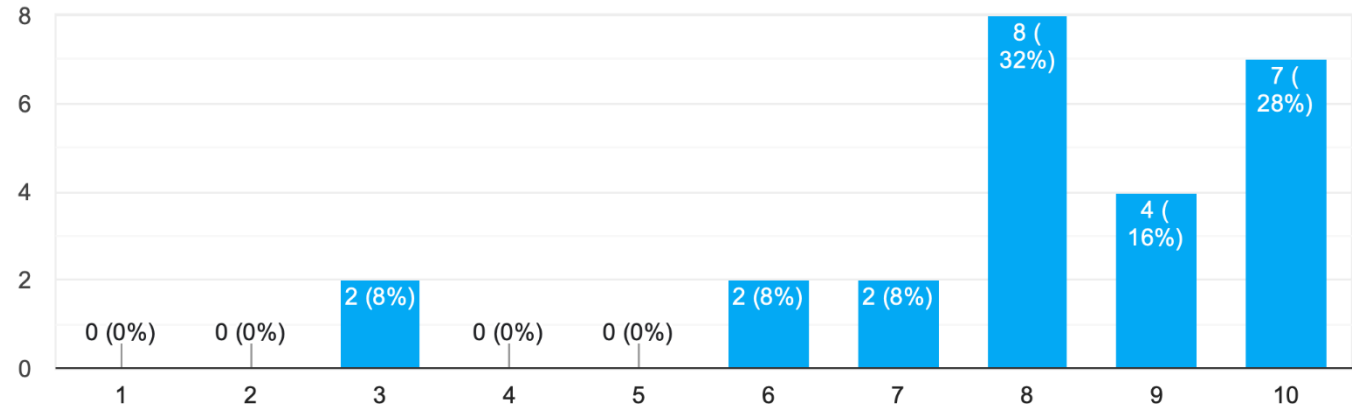
I thought there was too much inconsistency in this system.

25 responses



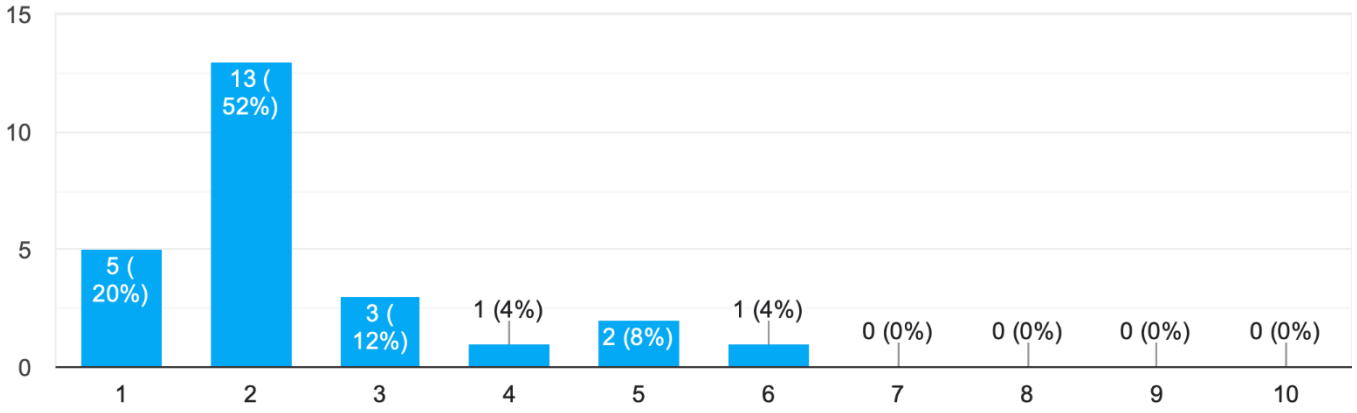
I would imagine that most people would learn to use this system very quickly.

25 responses



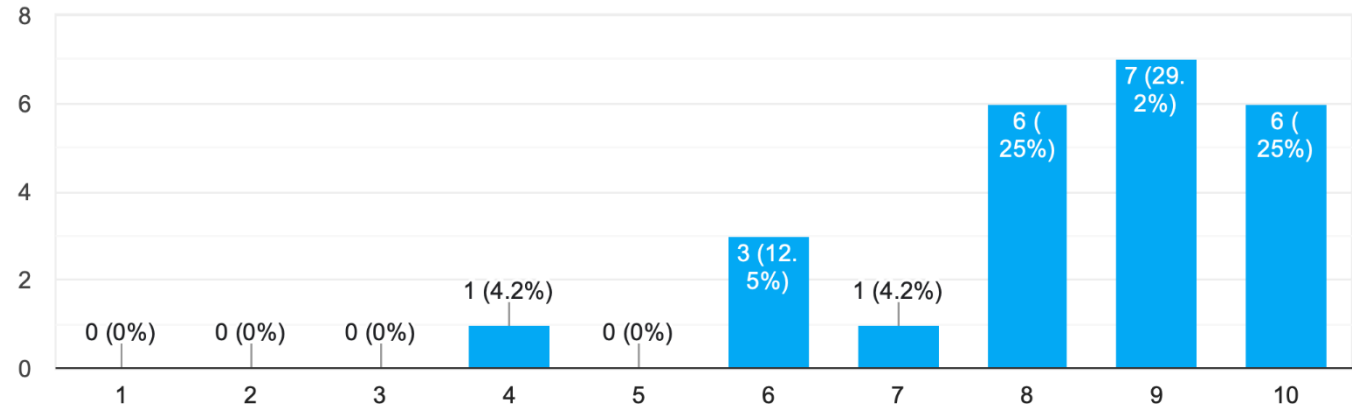
I found the system very cumbersome to use.

25 responses



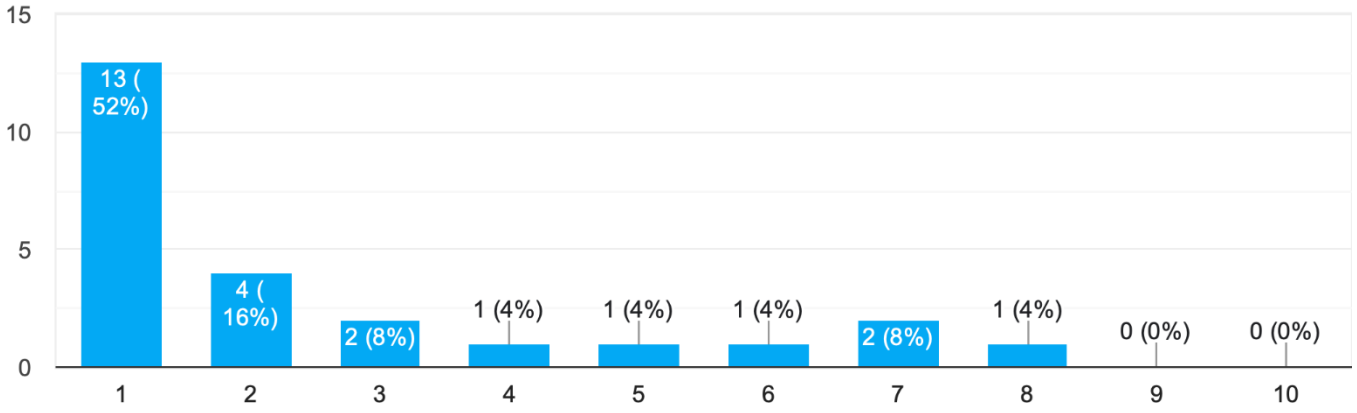
I felt very confident using the system.

24 responses



I needed to learn a lot of things before I could get going with this system.

25 responses



Additional Feedback:

1. What do you like the most about this website? (bold is the client's response)
   - Drag and drop stuff
   - simplicity
   - Drag and Drop
   - Well designed
   - Very responsive and communicates the functionality well - does what its supposed to do.
   - The drag and drop, and the custom backgrounds.
   - Admin edit manager system
   - I really like the simplicity of the main interface. Easy to understand
   - Drag and drop feature
   - The drag and drop simplicity of everything aas great. Felt professionally done.

- The drag and drop functionaly is very good
- Drag and drop feature and overall simple design
- Dragggggg
- I liked the drag and drop it was very smooth and easy to use
- The live updating
- The drag and drop functionality was very impressive
- The ease of the drag and drop system. Also makes visualizing the scheduling easy.
- Drag and drop
- **Ease of use and the man power projection feature.**
- **The ability to scale and predict further into the future then a limited space whiteboard offered.**

2. If you can change one thing about this website, what would you do? (bold is the client's response)
    - The calendar selection tool
    - fix about and other errors
    - Not much, really is well done
    - Honestly, nothing from my perspective, but I'm not a construction worker.
    - The visual hierarchy (adding jobs and employees should be more visible)
    - Have tooltips in buttons saying what button does what
    - adding employee to job, employee's profile photo is not shown on employee list
    - make it easier to assign a employee to multiple jobs
    - Creating a new manager account should be more intuitive and easier to navigate
    - Honestly nothing, worked like a charm
    - The manager/admin separation
    - Maybe more tooltip and user responsiveness
    - Nothing
    - Not much honestly did a really great job
    - Dark mode/
    - Make the profile picture of a new employee show up immediately after selecting
    - Simpler
    - **Thinking long and hard here... I would not change anything.**
    - **Minor quality of life things. The overall function is what we asked for.**
3. What is the one thing you wish the website could do that it doesn't already? (bold is the client's response)
    - N/A
    - Refresh for added user
    - im not sure
    - Small little tooltips for certain actions like archiving
    - All is well with me.
    - Can't think of anything
    - Indicate what projection is doing for the main page. Just some help stuff.
    - I know the website is meant for managers but it would be really cool for employees to see it
    - Cant think of anything
    - Have documentation and about sections
    - None
    - **Make coffee. It does all that we need.**
    - **Copy to clipboard**
4. Do you have any other feedback you'd like to share? (bold is the client's response)
    - Good work
    - good job (ˆ ∀ ˆ)

- Well done!!
- None.
- The functionality is great, I'd work on the aesthetics and hierarchy (keep in mind, the people using this won't be technologically adept)
- Looks great and the functionality works really well.
- Great design and all functionalities work very well, just some minor bugs but overral wonderful job!
- Nopr
- Nope!
- It's really good! Well done
- A bit of UI
- **The team was great to work with and very collaborative on options, features and implementation of the product.**
- **Great work team. Everyone that has seen the board in action so far has been blown away.**

Core requirements delivered/any requirements unimplemented/partially working

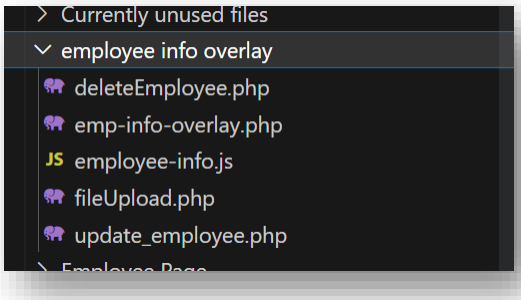
Please refer Table 1 in "Project Management" section

Known Bugs

- Closing any overlay will currently run ajax_search() (essentially refreshing the employees on the employee page and jobs on the job page), though change_active() (runs when changing the status of employees) also runs ajax_search(), essentially causing a double refresh whenever change_active() followed by closeOverlay() is ran (e.g. changing the status of an employee who is assigned to a job will cause a popup that will need to be closed but also runs change_active()). This only affects performance a little for some edge use cases.

Lessons learned

- We hosted a beta version of the website one of our personal computers, but that was not always running nor up-to-date. If we had set up continuous deployment earlier, where a beta version was hosted on their system that would auto-update, that would have allowed us to get more client feedback earlier.
- We never established a naming convention for functions, variables, and file names. Now our project is filled with camelCase, kabob-case, snake_case, Title Case, and Sentence case. This could be fixed with a linter.



- We did not figure out testing with phpunit early enough, leading a lot of our files without tests and untestable without some refactoring. This also goes against our "test first, then write code" process, which was unfortunately rarely followed. This resulted in a lot of files that are simply without tests, and as we are nearing project completion, not enough time to go back and make themT
- We did not have a front-end testing framework, which would have made code reviews a lot easier, as whoever is reviewing the code basically has to go through all the use cases that were touched as a result of the changes.

Luckily, the project isn't too massive, so this has not resulted in a significant time loss other than for the person reviewing the code.

- We never established branch protections on master, allowing for some unreviewed pull requests to be merged as well as a force push reverting changes. This in total probably cost us an hour or two.
- We did not have much or an incentive to promote project knowledge sharing and to reducing knowledge silos, allowing some group members to have significantly more project knowledge than others, reducing their efficiency

Project Continuation Guide

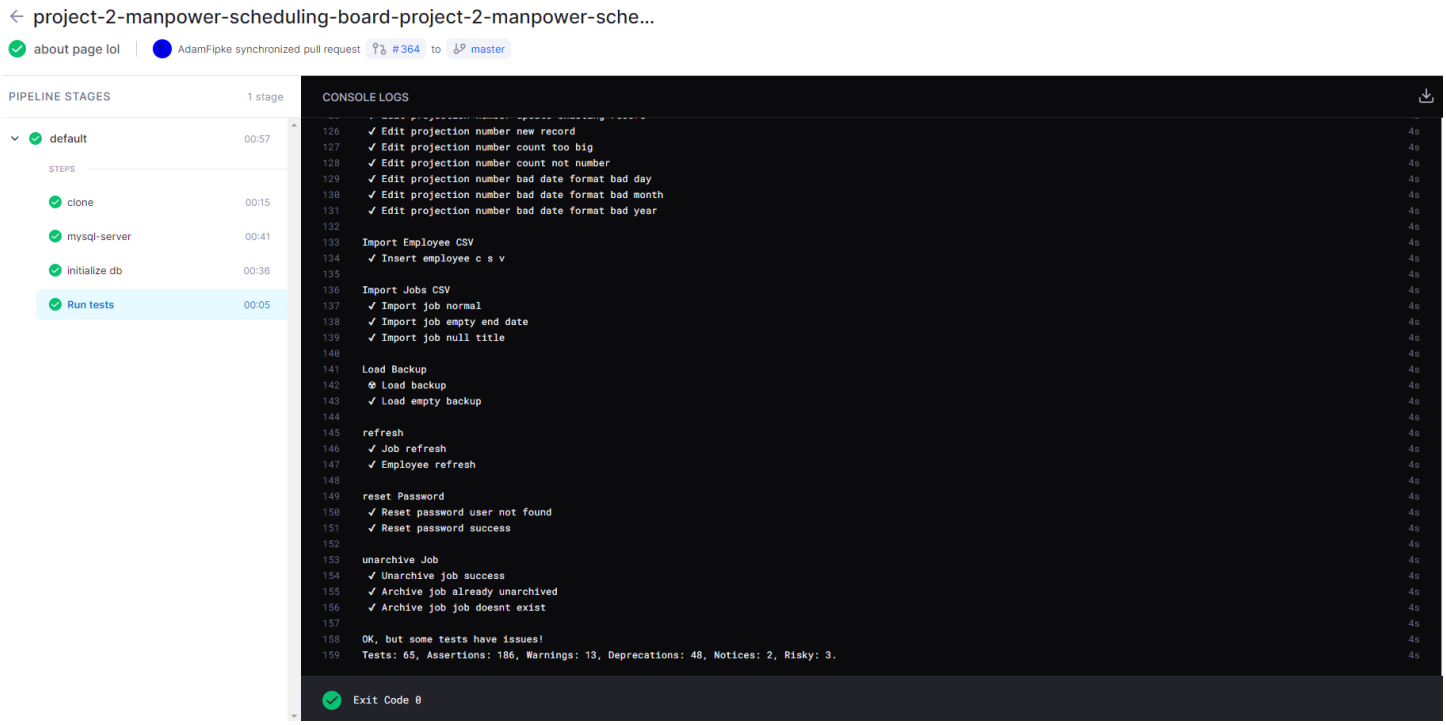
Workflow Environment

To get set up for this, simply follow the same steps in the installation details below. Once you get the docker container running and the website is viewable, any changes to the server files will automatically be reflected on the server. If there is any changes to the database, one will need to either alter the already existing database through phpMyAdmin or run *docker-compose down -v* (WARNING: THIS WILL RESET THE DATABASE) and *docker-compose up -d – build*

If there was changes made to the Javascript or CSS files, the changes might not be immediately reflected due to the browser caching those files, and one will need to press ctrl+f5 to force the cache to clear for the page and fix it.

Once changes are done and made, commit them and push them into the github repo (if working in a group, it is probably best to create a pull request first as opposed to pushing to master, and also make sure to write tests as well).

We used Drone in order to do Continuous Integration for this (i.e. auto-run the tests on all pull requests, allows everyone to see if the new code broke anything). The *.drone.yml* file has this workflow. One will likely need to set up their own Drone server to use the same workflow (we used our university's Drone server), though one could choose another Continuous Integration environment if need be.



An example Drone workflow result

Testing Environment

We used PHPUnit to run all of our backend tests. We do this by building a PHPUnit container with docker.

Steps to run tests:

1. Assuming you have already navigated to where you have clones the repo, run the command *docker build . -t phptest*

2. Next, to actually run the tests, run the command *docker run --network=project-2-manpower-scheduling-board-project-2-manpower-scheduling-board_default --rm -v //$(pwd):/app jitesoft/phpunit phpunit ./tests –testdox*

(you might need to adjust the command so it uses the proper container name).

Deployment Environment

While developing the project, we simply had one of us host the website on their own personal server, which allowed the client to sorta checkout and experiment with the system (the main limitation being that his computer was not always open).

When it came to actually deploying it on the client's computer, we simply followed the same installation details below (albiet it was a bit bumpy due to CPU virtualisation not being enabled in the server's BIOS and some temporary docker image issues, but we got it all sorted). The project it stored on the desktop of the client's computer, and within that folder there is another file called something along the lines of "sync project" which simply pulls any new changes from the deployment branch on the client fork.

The system is currently installed on a client's computer within their server room that is connected to their local physical network. Any of the computers connected to that network should be able to view the web server at *[the computer's ip]:8080* and phpMyAdmin at *[the computer's ip]:5000*. Any devices outside this network cannot access the page, making it very secure.

The computer's data is also backed up into the client's actual server, which is then backed up into the cloud (this is on top of our own backup system), making it triple redundant.

Where is code located?

Client fork link: https://github.com/AdamFipke/manpower-scheduling-board-client-fork

Installation Details

Steps:

1. Clone this repo to wherever you'd like (assuming you have git installed)

2. Open a terminal and navigate to where you cloned the repo

3. Run the command *docker-compose up -d --build* (assuming you have docker installed)

4. Head to http://localhost:8080/ to check out the website! Can also go to http://localhost:5000/ to use phpMyAdmin (might need to swap out the localhost part with an IP, given by client)

User Manual/User Documentation

There's a help page(all user) and documentation page(admin user) on the actual website. Here are screenshots of it. Documentation page is just a simplified version of this report. Help page has 2 sections of the documentation content. All feature related demonstrations and database demonstration are displayed as gifs, with click to start/stop.

Documentation page:

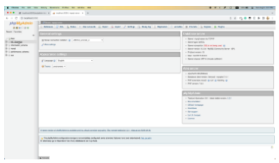
HORIZON ELECTRIC INC. | Documentation | Functions | User View → | Logout

Welcome to this Scheduling Board! This user documentation will guide you through the various functions and features. Below are the key sections to make most of your experience. Click 🔍 to enlarge gifs/pictures. Click original gifs to start original gifs.

1. Scheduler Features (You may want to check it out)
2. Troubleshooting (You may want to check it out)
3. Admin's Functions
4. Contact and Support
5. Getting Started
6. Managers' Functions

Scheduler Features

The scheduler saves any modifications made during the day at midnight.
The scheduler runs under a Docker container. What is Docker and its container: Docker allows you to create and run containers. Containers are like small packages that contain everything need to run an application, such as the code, the libraries, the settings, and so on. Containers are useful because they make it easy to move application from one computer to another without worrying about compatibility issues or installation problems. Here is a video explains Docker.
Database is available at: http://localhost:5000 (replace localhost with your static ip), then click "HE_Schedule" in the right.

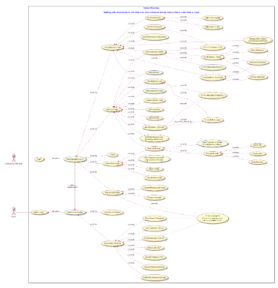


Database 🔍

The scheduler uses port 5000 for phpmyadmin(phpmyadmin is a software tool to manage a database) and port 8080 for the website. If any conflicts arise, these ports can be changed. In all demonstration, the develper used different port numbers than what deployed in the office.
The website developed using HTML, CSS, JS, PHP, and MySQL as the primary programming languages.
The developers validate user login with frontend and backend validation and encrypt passwords via hash before storing them in a database.
The website is hosted on a local server that does not have access to the internet, therefore no web-based APIs on the server may be used.
Use case flowchart below. It defines particioants(manager and admin in this case), as well as their roles and interactions.

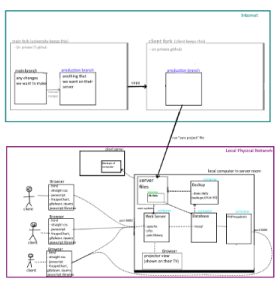
HORIZON ELECTRIC INC. | Documentation | Functions | User View → | Logout

Use case flowchart below. It defines particioants(manager and admin in this case), as well as their roles and interactions.



Use Case Flowchart 🔍

Architecture diagram below. This is a high level overview of the main components of the system and how they connect.



Architecture Diagram 🔍

Database design is as follow. This is a high level diagram that describes how information flows between different components of the system.

ER Diagram

Data Flow Diagram Level 0

Data Flow Diagram Level 1

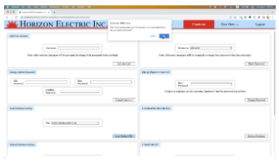
Troubleshooting

If you encounter any issues, you can try restarting the Docker containers using the following steps:
In terminal, navigate to the project folder (here is a wikihow link instruction).
Type "docker compose down -v" to stop Docker container. **Warning: Caution! THIS WILL RESET THE DATABASE!** Make sure to download a backup first!
Type "docker compose up -d" to restart Docker container.
If you need to restore database, instruction is provided in Admin Managers' Functions section at Load database backup.

Admin's Functions

In addition to manager's functions, the admin manager has extra functions under the Functions tab. The followings are the main features:

Database and Backup Related Functions



Load Database Backup

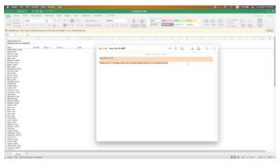
Download Backup Database

Upload Database Backup

Import CSV

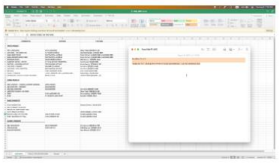
HORIZON ELECTRIC INC. | Documentation | Functions | User View → | Logout

Import CSV



Import Employee CSV

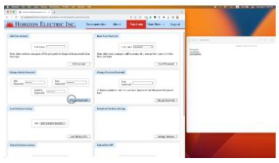
This function should only be done once.
CSV format to follow: 3 lines of header and 3 lines of footer



Import Job CSV

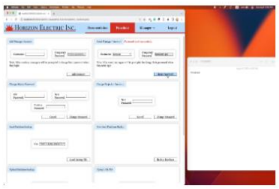
This function should only be done once.
CSV format to follow: 3 lines of header and 3 lines of footer

Other Functions

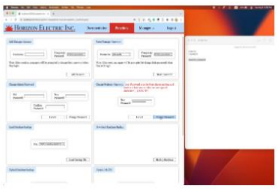


Change Admin Password

Note: this change password function goes more security check than change manager password



Reset Manager Password



Change Projector Password

Create Account

Contact and Support

Please contact Next Generation Computers Martin(He/Him/His: deployment/IP address, etc) at Martin@ngcit.ca and/or Adam Fipke(He/Him/His: project developer; EMERGENCY ONLY) at adamfipke@gmail.com

Getting Started

Access the Website: use Chrome or Edge for optimal performance
Create Account: only admin manager can create new account

Managers' Functions

The followings are the main features.

Job Related Functions

Add Job

Archive Job

Unarchive Job

HORIZON ELECTRIC INC. | Documentation | Functions | User View → | Logout

Show all jobs: archived jobs and job in Jobs Tab listings.

Edit Job

Edit number of employees for job month.

Notice: you can enter a number or click/hold the up/down button

Delete Job

Employee Related Functions

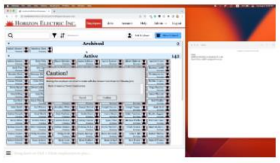
Add Employee

Edit Employee Details

Move employee to a job - drag to green zone

HORIZON ELECTRIC INC. | Documentation | Functions | User View → | Logout

Batch Assign



Move employee to archived: employee with assigned job will show a popup message

Edit employee card to archived

Move employee to inactive

Move an employee to multiple jobs at the same time - will show as duplicated

Move unnasigned employee through employee list

HORIZON ELECTRIC INC. | Documentation | Functions | User View → | Logout

Move Employees Through Different Windows

Note: this is not guaranteed to compatible (just a bonus feature)

Unarchive Employee: by dragging or by editing an employee card

Order and Search

Custom Order

Order Job

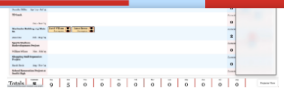
Order Employee

Search Employee: through employee list or Employees Tab

HORIZON ELECTRIC INC.

Documentation | Functions | User View → | Logout

Search Job



Search Employee: through employee list or Employees Tab

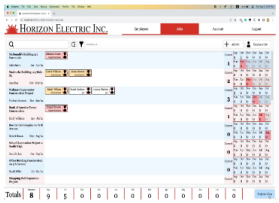
Projector View

Login as Projector View User

Admin can change password

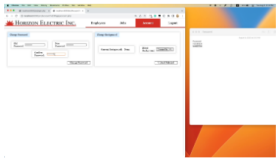
Username: projector

Password: projector_password



Go to Projector View

Other Functions



Change Password

Change Background

Help page:

HORIZON ELECTRIC INC.

Employees | Jobs | History | Account | Help | Logout

Welcome to this Scheduling Board! This user documentation will guide you through the various functions and features. Below are the key sections to make most of your experience. Click ⌕ to enlarge gifs/pictures. Click original gifs to start original gifs.

1. Getting Started
2. Managers' Functions

Getting Started

Access the Website: use Chrome or Edge for optimal performance
Create Account: only admin manager can create new account

Managers' Functions

The followings are the main features.
Job Related Functions

Add Job

Archive Job

Unarchive Job

Batch Assign

Move employee to archived: employee with assigned job will show a popup message

Edit employee card to archived

Move employee to inactive

Move an employee to multiple jobs at the same time - will show as duplicated

Move unnasigned employee through employee list

HORIZON ELECTRIC INC.
Employees Jobs History Account Help Logout

Show all jobs: archived jobs and job in Jobs Tab listings.

Edit Job

Edit number of employees for job month.

Notice: you can enter a number or click/hold the up/down button

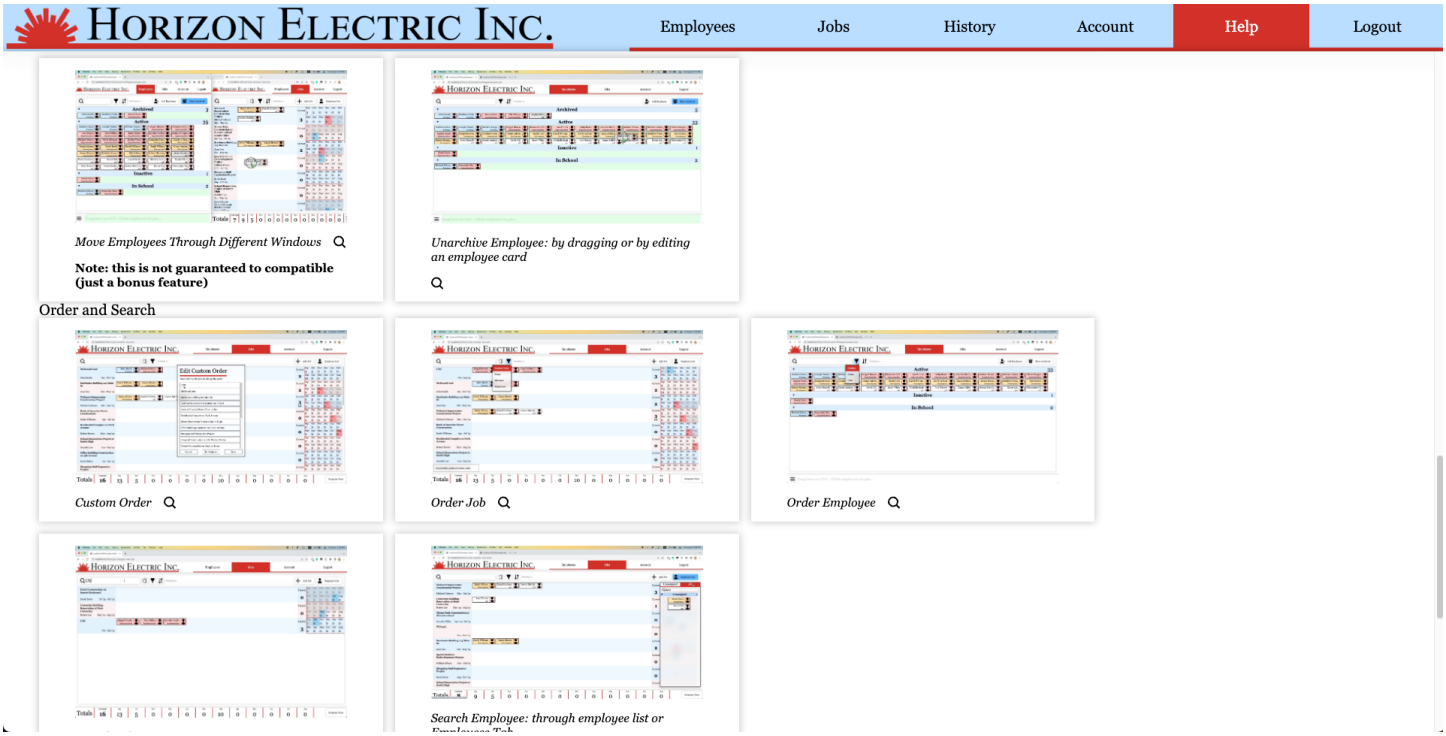
Delete Job

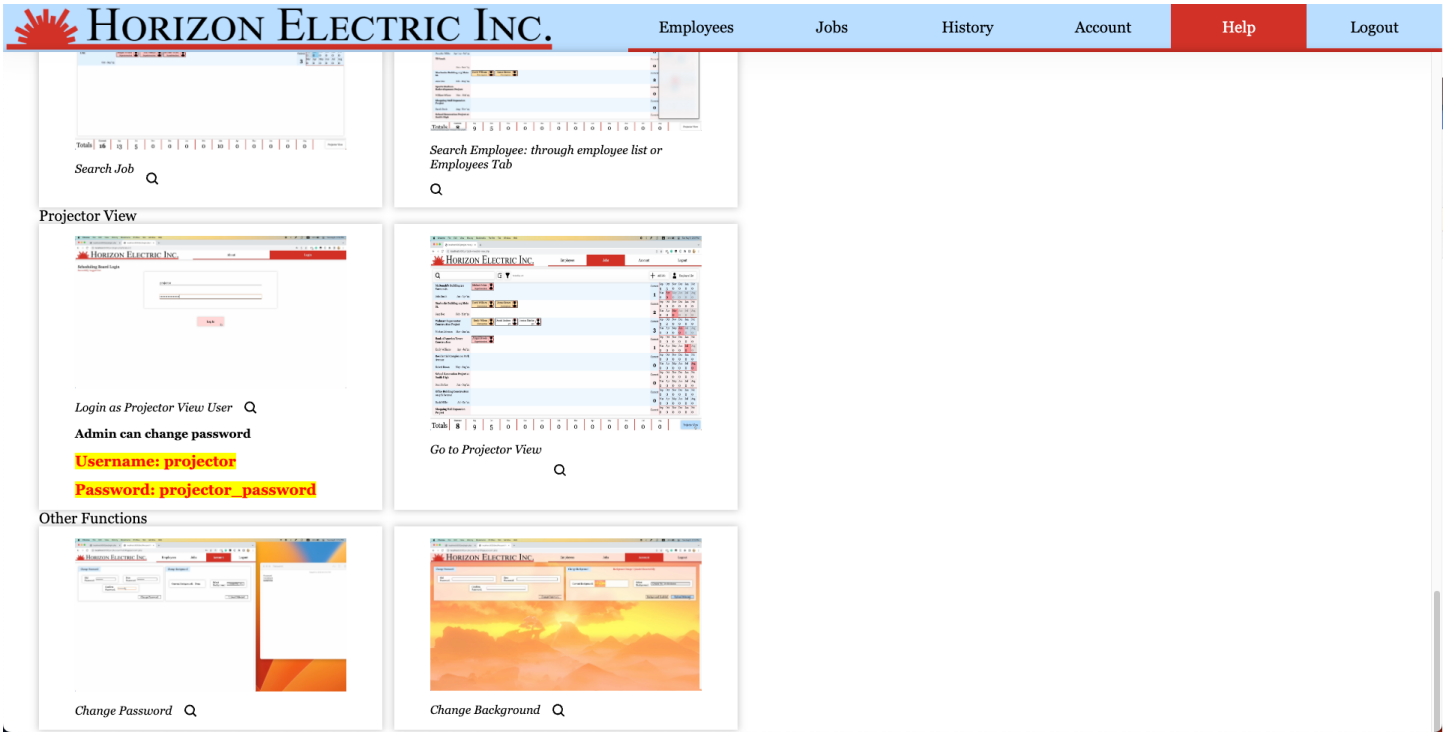
Employee Related Functions

Add Employee

Edit Employee Details

Move employee to a job - drag to green zone





Tests Needed

- We have no frontend testing framework, adding one would allow a lot more to be tested (such as the many generators, the ones that download/upload backups, or the upload image ones (PHPUnit doesn't like the compression library we used))
- A lot of more backend ones like: mass_remove(), change username, auto refresh, archive employee, change employee active status, etc.
- Load backup needs to be fixed so that it resets to how the database was before the function was ran.
- Change background and employee image successful upload (again, compression library issues)

Potential New Features To Add

- Some optimizations to only update specific parts of the jobs/employees page when an employee moves as opposed to the whole thing (e.g. only refresh the two jobs being updated when moving an employee from one job to another, as opposed to all jobs)
- store assignment data for who is inactive/in school and for how long (similar to jobs), which can then be graphed
- add a feature that lets managers set a date for when an employee is no longer inactive/in school, where it will auto make them active again
- Change how graphs are generated so that there is only one instead of generating one for each job
- Ones from our kanban board:

1	Salt passwords #366		Nice to Have
2	View Diagnostics #67		Nice to Have
3	Undo #56		Nice to Have
4	Dark mode #145	JordoRob	Nice to Have
5	Edit Employee Front-End Validation #240		Nice to Have
6	Pin job #149		Nice to Have
7	Show unassigned employees on employee page #334		Nice to Have
8	Remove redseal from database and refactor inserts/updates/queries/etc #156		Nice to Have
9	Change password client-side validation #305		Nice to Have
10	Customise font - maybe don't do #351		Nice to Have

Feature not implemented

Please refer Table 1 in “Project Management” section.