



WPI

7Factor Staffing Tool

A Major Qualifying Project submitted to the faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Science.

In cooperation with 7Factor Software, LLC

Submitted March 6, 2022

By:

Dyllan Cole

Noah Olson

Andrew Whitney

Evan Llewellyn

Project Adviser:

Professor Joshua Cuneo

This report represents the work of WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the projects program at WPI, please see

<http://www.wpi.edu/academics/ugradstudies/project-learning.html>

Abstract

7Factor is a software development company based in Atlanta, GA. Operating under the software consultant model 7Factor collaborates with clients to provide software solutions which fit their clients' needs. This business model relies heavily on keeping track of employees' billable hours. This is the backbone of their company, and having a reliable, efficient, and maintainable application that allows them to do this tracking was the objective of this Major Qualifying Project (MQP). Throughout the project, we refined and improved upon the platform created by our predecessors, focusing on an elegant user experience and providing analytical tools, which will allow 7Factor to optimize their profits and their employees' workload. Our development efforts also included back-end changes to support new features and improve security.

Acknowledgements

For assisting us with our work on this project, and making this project possible to begin with, we would like to thank the following people:

Professor Joshua Cuneo (Project Advisor)

Jeremy Duvall (7Factor CEO)

Additionally, we would like to thank the members of the first MQP group to work on this project for creating the foundation upon which our work was built:

Panagiotis Argyrakis

Rafael Pimentel

Nicholas Wood

Table of Contents

Abstract	2
Acknowledgements	3
Table of Contents	4
List of Figures	6
List of Tables	8
Executive Summary	9
About 7Factor	9
Time Tracking and Current Software	9
Design and Implementation of the New Solution	10
Results	10
Future Work	10
Introduction	13
Background	14
7Factor	14
TimeIQ	14
Previous Group's Accomplishments	15
Previous Group's Future Work	16
User Stories	17
Methodology	20
Team Methodology	20
Agile	20
Development Tools	21
Gantt Chart	21
Trello	21
GitHub	22
Software Libraries	22
Back-End	23
Front-End	24
User Interface Mockups	25

Results	28
User Interface: Before & After	28
Login Page	28
Time Entry Page	30
Rate Card Page	31
Project Page	35
Employee Page	36
User Interface: Additions	39
Back-End Changes	42
Role-Based Authentication	43
Recommendations and Future Work	46
Conclusion	49
Bibliography	50
Appendix	51
A. Development Environment Setup	51
B. Running the Staffing Tool	51
Please Note	51
Instructions	51

List of Figures

Figure 1: Mockup for the home screen

Figure 2: Mockup for timesheet screen

Figure 3: Mockup for employees screen

Figure 4: Mockup for dialog to create a project

Figure 5: Previous team's login page

Figure 6: New login page

Figure 7: Previous team's time entry page

Figure 8: New time entry page

Figure 9: Previous team's adding rate card screen with only one role

Figure 10: Previous team's adding rate card screen with multiple roles

Figure 11: Previous team's adding rate card screen with title overlapping other fields

Figure 12: New rate card page

Figure 13: Each rate card can be edited and created using this dialog box.

Figure 14: Editing a position on a rate card

Figure 15: Viewing past rate changes for a position

Figure 16: Previous team's project creation screen

Figure 17: New project management screen

Figure 18: Dialog to create project

Figure 19: Dialog to edit project

Figure 20: Previous team's employee creation screen

Figure 21: Previous team's people screen with no employees created

Figure 22: Previous team's people screen with employee present and fields filled

Figure 23: New employee page

Figure 24: Dialog to create employee

Figure 25: Dialog to edit employee

Figure 26: Home Screen displaying information about the logged-in user

Figure 27: Weekly view of analytics page

Figure 28: Overall view of analytics page

Figure 29: Staffing Tool Database Entity-Relationship Diagram

Figure 30: Roles on Auth0

Figure 31: User with Roles on Auth0

Figure 32: If a user does not have access to part of the application, this screen will be displayed.

List of Tables

Table 1: Database Access Design Pattern

Executive Summary

About 7Factor

7Factor is an Atlanta-based software development company with about 55 employees. It specializes in developing cloud architectures, continuous delivery pipelines, and custom systems and application development. 7Factor has developed software solutions for big-name companies such as Delta, Home Depot, and Cox Automotive using the software consultant model. Under this model, 7Factor provides end-to-end project work, dedicated bug-fixing retainer teams, and consulting for complex issues.

Time Tracking and Current Software

As a software engineering consultancy, 7Factor needs a robust time-tracking system that allows them to track hours for many employees working on many different projects. Time-tracking may seem like a simple concept, but as our group learned in implementing our project, there are many considerations that are anything but simple: different customers may be billed different rates, different employees are paid different amounts, employees can have hours that are not attached to a specific project, some employees may be billed at different rates only for specific projects, among other considerations. Currently, that role is fulfilled by a paid commercial solution called TimeIQ. As 7Factor has grown, however, they have found that TimeIQ is requiring far too much manual work and is not a very good fit for their use-case. This is in addition to the obvious benefit of not having to pay on a per-employee basis to use an in-house solution, as TimeIQ charges \$5 per employee per month, which rapidly adds up as 7Factor grows.

Design and Implementation of the New Solution

Since this is a continuation of a previous team's project, the overall design was already decided upon. The design maintains the concept of a project, client, employee, and rate card; however there is one key change that has been made in this area: an employee can be assigned a rate card and a position on that rate card as well as a default salary. If no position is assigned to the employee, and they are assigned to a project, the payment calculation will use their default salary, but if they do have a position for that project assignment, that position's hourly rate will be used for payment calculations instead. We also changed the landing page to be a home page with simple analytics instead of the time entry screen. We did not change the software being used to build this application, which is PostgreSQL for the database, the Spring Boot Java framework for the REST API, and React for the front-end.

Results

Within the timeframe of our project, we created new user interface mockups to redesign the front-end, implemented them in React, and added and modified back-end entities as our project evolved. We also improved upon the previous team's Auth0 authentication strategy by adding in roles for users so that a manager can control who has access to certain parts of the application. 7Factor's biggest request for us to implement during our MQP, however, was analytics. In line with that feature request, we implemented financial analytics, project analytics, and employee analytics so that the complex data handled by the staffing tool can be visualized to understand financial performance at a glance.

Future Work

Though we did greatly expand the functionality of the staffing tool during our project, it is by no means complete. Towards the end of our project, we identified several

areas that will require future work before the staffing tool is used in production. These are, in order of importance, server-side role-based authentication, data transfer objects (DTOs), and responsive design. None of these were specific feature requests from 7Factor within the scope of our project work, which is why we did not specifically implement these features.

In terms of server-side role-based authentication, the next group essentially just needs to validate Auth0 roles on the server-side. Currently, parts of the UI are restricted based on Auth0 roles so that you can only enter or modify data that you have the corresponding role for. The back-end does not verify these roles, however, and therefore it is possible that someone could simply use their valid Auth0 token to manually send a request that they should not be able to. We did not deem this a major problem, as someone would need to have first had a valid account created for them by 7Factor. Therefore, we decided to leave this for a future group so we could focus on our core features.

A major shortcoming of our front-end right now is the way it sends and receives requests to and from the back-end. When a request is made, the back-end simply responds with a directly serialized representation of its internal data. This means that, for instance, if you request an employee's data, you might also get information on all of said employee's time entries due to the back-end association. This is not a problem when you are dealing with a small amount of data, but this could rapidly balloon in a production deployment dealing with a lot of data. The solution to this problem is implementing data transfer objects (DTOs), which only contain the information necessary to a specific request. By implementing DTOs and adding more specific API endpoints, the amount of data that needs to be sent back and forth between the front-end and back-end could be greatly reduced.

Finally, we believe that we have produced a satisfactory desktop user experience, but the front-end could greatly benefit from the implementation of responsive design principles. Right now, it looks quite weird on a phone, tablet, or nonstandard display. Using responsive design, however, the same front-end that we already have could be

used to bring all of our functionality to any device. This would greatly enhance the convenience of using the staffing tool: employees could enter their hours on the go from their cell phones, and managers could quickly check in on the financial performance of a project from their tablet. This is mainly an issue of styling. All the functionalities would remain the same, but styles would need to be adjusted based on screen size so that elements can dynamically scale to accommodate a wider range of screen sizes.

Introduction

7Factor, a software engineering company, located in Atlanta, Georgia, was looking for a new solution to TimeIQ, which is a time tracking software application for teams. Through interviewing Jeremy Duvall, Founder and CEO of 7Factor, it was apparent that TimeIQ was not modular enough to accommodate the needs of his company. 7Factor acts as a consultant, where they design custom DevOps and other cloud-based products for other companies. As a result, they need to bill these companies which can lead to complex time keeping, all with varying rates, projects, and employees. 7Factor is keen on being able to bill different rates to different clients based on the project and skill sets required.

Thus, 7Factor wants to build their own Software as a Service (SaaS) that is similar to TimeIQ but is able to handle the complexities of their time-keeping and billing needs. That is the purpose of this MQP: to build the staffing tool software that they want. This new tool is designed to overcome the main challenges they currently face with TimeIQ, which are an overcomplicated user interface and the inability to set a custom billing rate for each employee/project pairing individually. By solving these issues, 7Factor hopes that this new staffing tool will be able to handle their business structure and make it easy for the user to use.

Background

7Factor

7Factor is an Atlanta-based software development company with about four dozen employees. It specializes in developing cloud architectures, continuous delivery pipelines, and custom systems and application development. 7Factor has developed software solutions for big-name companies such as Delta, Home Depot, and Cox Automotive using the software consultant model. Under this model, 7Factor provides end-to-end project work, dedicated bug-fixing retainer teams, and consulting for complex issues.

TimeIQ

7Factor uses the staffing management tool TimeIQ to organize its clients, their projects, and the employees who work on them. At the core of TimeIQ are its time tracking sheets, which allow employees to log the amount of time that they spend on specific projects. This data is then used by TimeIQ to generate reports describing hours spent per person on each task and the subsequent cost to bill clients. Additionally, TimeIQ offers budgeting insights and can estimate profits.

TimeIQ's limitations have caused 7Factor to search for a more tailored staffing management tool. Notably, TimeIQ does not support billing different rates per project. 7Factor has enabled this functionality by creating new "services" for each rate. However, this workaround requires developers to remember their own rates. With 7Factor's roughly 55 developers, this strategy leads to inaccurate reporting and unnecessary expenditure. Overall, while TimeIQ adequately meets 7Factor's needs, a customized platform would streamline workflow and save valuable work hours.

Previous Group's Accomplishments

This MQP is part of a multi-year effort to build 7Factor a fully working staffing tool. The previous team primarily focused on laying the back-end foundations for 7Factor's staffing tool platform. They designed a database schema to represent the relations between projects, assignments, employees and rate cards and implemented a multi-layered design pattern to process and execute operations requested by the front-end. This strategy is represented by table 1.

Repository	Accesses data from database
Service	Validates request attributes, handles errors
Controller	Handles requests from front-end, sends responses

Table 1: Database Access Design Pattern

At the most basic level, this pattern enables creating, reading, updating, and deleting (CRUD) employees, projects, and rate cards. The previous team additionally added functionality to assign employees to projects and to access and submit time entry data. Lastly, to accommodate the unique business model of 7Factor, the previous team designed its back-end (schema and queries) to handle scheduled salary adjustments.

To ensure the functionality of back-end features throughout the development process, our predecessors applied two different integration testing methods. The first method uses sliced WebMVC mocked calls to instantiate only part of the application and make (and verify the results of) http requests to controllers. The second method

uses Docker to create a PostgreSQL container and tests the database itself. These two strategies ensure that the back-end is thoroughly tested.

While Auth0 integration was touted as a key feature of the previous team's application, we found that it was implemented improperly. In the state that we received it, Auth0 was not interacting with the back-end and was throwing errors. We spent some of our initial development time getting this up and running.

Front-end features were much more limited than the back-end, and many bugs impaired the user experience. The previous team created a login page, time entry page, and pages to add employees, rate cards, and projects. Pages were designed using React and Material Design but lacked detailed styling. Additionally, the interface did not provide a way to view created projects, rate cards, or submitted time entry sheets. Consequently, we identified the front-end as our team's primary development focus.

Previous Group's Future Work

In their MQP report, the previous group to work on this project detailed several next steps. Some of these were items that they had initially planned to do themselves but then ran out of time to implement, and others were features that they thought would improve the overall experience:

- Front-end support for editing rate cards, employees, and projects
- Restricting projects listed on time entry page to only the ones that the employee is actively working on
- Integrating the front-end and back-end with 7Factor's continuous integration pipeline
- Representing clients as a database object and adding front-end handling for that
- Allowing lazily-fetched objects on the front-end to increase performance
- Adding a search feature to the employee and project views

At the start of our project, we reviewed the previous MQP report and heavily leaned on the previous group's future work section in planning our initial tasks. In

particular, the front-end support for editing rate cards, employees, and projects was one of our first major changes. As the requirements and goals of this project shift somewhat over time, we did not implement all of the recommendations of the previous group. For instance, we did not deem lazily-fetched objects on the front-end to be a particularly high priority item. We acknowledge that all of the future work items would be good additions to the project, but we prioritized those which had a tangible impact on the functionality and user experience.

User Stories

The previous group also provided us with a list of initial requirements for the 7Factor staffing tool platform. These are the requirements that they received from Jeremy when working on the project with him and are thus also requirements that we have actively worked towards. The core requirements were as follows:

1. As an administrator, I want to invite new consultants to the platform so they can create accounts.
2. As an administrator, I want to create a project along with its client, start date, end date, and set of default billing rates.
3. As an administrator, I want to assign a consultant to a project along with the hourly rate at which they will be billed to the client.
4. As an administrator, I want to remove a consultant from a project.
5. As an administrator, I want to view the total billable hours on a project for a specific time range so I can bill the client.
6. As an administrator, I want to set the yearly salary of a consultant and specify which date the change will take effect on.
7. As an administrator, I want to correct an existing time entry on behalf of a consultant.
8. As a consultant, I want to log the hours I spent on each project, on overheads, or on the bench for a specific week.

9. As a consultant, I want to request to modify an existing time entry so I can correct a mistake on a time entry that has already been approved.

In addition to the core requirements, the previous group listed out several stretch requirements to be completed as time allowed. They were as follows:

1. As an administrator, I want to see a graph of the total expenses versus the revenue of the business so I can better understand the profitability
2. As an administrator, I want to see a graph of the expenses versus the revenue for a specific project so I can better understand the profitability of a project.
3. As an administrator, I want to archive a finished project.
4. As an administrator, I want to view archived projects.

When we were handed the project, we assessed its state and concluded that the previous group completed core requirements 2 and 3 and did not complete any of the stretch requirements. It should be noted, however, that these are overall requirements for the staffing tool itself, not necessarily for the previous group's work on the staffing tool. In the course of our work on the project, the requirements for the staffing tool have grown. With those additions, here are all of the core requirements as they now stand:

1. As an administrator, I want to invite new consultants to the platform so they can create accounts.
2. As an administrator, I want to archive consultants so that they can no longer access the platform.
3. As an administrator, I want to create a project along with its client, start date, end date, and set of default billing rates.
4. As an administrator, I want to assign a consultant to a project along with the hourly rate at which they will be billed to the client.
5. As an administrator, I want to remove a consultant from a project.
6. As an administrator, I want to view the total billable hours on a project for a specific time range so I can bill the client.

7. As an administrator, I want to set the yearly salary of a consultant and specify which date the change will take effect on.
8. As an administrator, I want to correct an existing time entry on behalf of a consultant.
9. As an administrator, I want to restrict which parts of the staffing tool each consultant has access to dynamically.
10. As a consultant, I want to log the hours I spent on each project, on overheads, or on the bench for a specific week.
11. As a consultant, I want to request to modify an existing time entry so I can correct a mistake on a time entry that has already been approved.

As for the stretch requirements, we have kept all of the previous group's stretch requirements and also added some of our own. The stretch requirements with our additions are as follows:

1. As an administrator, I want to see a graph of the total expenses versus the revenue of the business so I can better understand the profitability
2. As an administrator, I want to see a graph of the expenses versus the revenue for a specific project so I can better understand the profitability of a project.
3. As an administrator, I want to create a client that projects can then be associated with.
4. As an administrator, I want to archive a finished project.
5. As an administrator, I want to view archived projects.
6. As a consultant, I want to select the theme of the staffing tool UI.

Methodology

Team Methodology

Agile

We did not strictly follow every part of the agile methodology to the letter, but we attempted to incorporate its iterative spirit into our team methodology. Though we did not have fixed-length sprints, we did have cycles that resembled sprints where we would meet with Jeremy to update him and gather new requirements, and we would then implement them and meet with him again. Another key integration of the agile methodology into our work was by always trying to accommodate changing requirements to produce a better end product. In the beginning, we did not always necessarily understand what we needed to implement with perfect clarity, so we sometimes had to change requirements ourselves when we understood them better. Additionally, we would sometimes get requirements for existing features changed, added, or removed when we brought them to Jeremy for feedback.

In the spirit of agile development, we tried to stick to a schedule of two scrum meetings a week for B21 and C22 terms. At the start of our project in A21 term, we had a haphazardly thrown-together meeting schedule that shifted around a lot; this was not at all conducive to us producing good work. We realized that we needed a more concrete schedule with frequent meetings, as team members were frequently getting stuck or not having enough work to do between meetings. After we started having regular scrum meetings, we found that both of those problems were resolved. Since we met regularly, it was easy to clear away any blocks on work that team members had. We also feel that regular scrum meetings were beneficial in keeping every member of the team up to date with what every other member was doing. Before we had regular scrum meetings, we would often go off and work on these in relative solitude without other

team members knowing much about what was going on with that work item until it was done. With the regular meetings, however, we found that we had a far better idea of what progress was being made on each feature, even if an individual team member had not worked on that feature themselves.

Development Tools

To streamline the development process, our team utilized a Gantt chart, Trello, and GitHub. In the following sections, we detail our usage of a Gantt chart, the basic functionality of these applications, and how we incorporated this functionality into our workflow.

Gantt Chart

The Gantt chart was used to visualize our requirements over the course of the project. It was where we would set up due dates and keep track of our progress on bigger tasks. We would use this chart to share our progress with Professor Cuneo and to plan our work for each term. As our timelines shifted, we revised our Gantt chart to ensure feasibility.

Trello

At the most basic level, Trello is a to-do list. Users can create boards on a per-project basis and use the Kanban-style columns to visualize team workflow. In our team's case, we created one board for our staffing tool application and five status columns. These included to-do, report to-do, doing, code review, and done. As a task nears completion, it moves from left to right across these columns. Additionally, Trello provides the user many ways to label tasks. We used labels to assign tasks to different team members, to qualify the type of work (e.g., bug fix, UI change, or API change), and to specify due dates. This labeling strategy helped add accountability to tasks and ensured that work was equally shared between team members. Overall, Trello helped our team organize our tasks on a granular level and ensured continual development.

GitHub

GitHub offers remote Git repository storage and a suite of version control features. Our team's code is hosted on Github and is split across three repositories: front-end, back-end, and database. GitHub provides an interface for viewing repo branches and commit history. Additionally, GitHub enables continuous integration by adding branch protections. Pull requests can be opened to merge in changes to protected branches after a review. Our team used pull requests when we wished to update our "develop" and "master" branches. By requiring additional eyes on code changes, this strategy minimized bug creation. Lastly, GitHub improves repository security by enabling GPG commit signing. This guarantees to 7Factor that our commits came from our team, and that they were not altered in transit. Ultimately, GitHub served as a host for our remote repos, prevented bugs, and enhanced the security of our application.

Software Libraries

Outside of niche or legally restricted environments, there are very few pieces of modern software that do not rely on external libraries to relieve the burden of implementing common functionality. With modern web applications, however, you essentially have two distinct environments relying on separate sets of libraries: the front-end, which runs in the browser and handles the user-interface, and the back-end, which receives API calls from the front-end and processes them to retrieve and update data. There is a major restriction on the environment of the front-end, being that until the recent addition of support for WebAssembly, which we will not be touching on, browsers only supported running JavaScript. Due to this, frameworks such as Node.js were developed to allow developers to write their back-end in JavaScript so that they could use one language for everything. Node.js significantly ate into the popularity of other frameworks, such as Ruby on Rails and PHP, but there are still many alternatives around.

Back-End

In our project's case, the previous team chose to use the Spring Framework for the back-end. Spring is a set of libraries written in Java that serves to simplify many mundane programming tasks but are particularly geared towards building web applications. The main library serving as the basis of our back-end is Spring Boot, which is geared towards helping developers quickly set up robust Spring applications with minimal configuration. As a library written in Java, Spring can be used by any language that runs on the JVM (Java Virtual Machine), including Java, Scala, Groovy, Kotlin, and many others. Our back-end is written in Java, as that is the most commonly known--and original--JVM language towards which most of the Spring documentation is geared.

Though we have described web applications as consisting of a front-end and a back-end environment, one also needs somewhere to store the data upon which the application is based. In many cases, a web application might be simple enough that there is not even a significant amount of data to manipulate, or the data can simply be stored in ordinary files. For web applications like ours, however, where there is a significant amount of inter-connected data to store and manipulate, you must use a database. We are using PostgreSQL as our database, which is a traditional SQL (Structured Query Language) database that has been around since 1996 and is currently a very popular choice with developers, ranking as the fourth most popular Database as of writing. Another recent trend with web applications is to use NoSQL (Not only SQL) databases, which feature different ways of storing and interacting with data. For the staffing tool, however, the data is highly structured and uniform, so a SQL database is very well suited to our use-case; NoSQL databases can be simpler to interact with and oftentimes feature better support for unstructured data, but this often comes with the drawback of reduced performance, so a NoSQL database would provide us with no real benefit.

Front-End

As mentioned, the front-end is more or less restricted to using JavaScript as a language, but there are a veritable cornucopia of JavaScript libraries to simplify website development. We used a framework called JSX (JavaScript Syntax Extension) that allows us to lay out and style the staffing tool by directly writing HTML and CSS in our JavaScript code. This functionality is provided by our main front-end software library, React. React is one of the most popular libraries for building reactive user interfaces in JavaScript. It allows us to easily lay out our website directly in our JavaScript code and dynamically update anything easily. The concept of dynamically updating our user interface based on our data is known as reactive programming, and is currently the most common paradigm used to develop data-heavy web applications.

Though React provides you with all the facilities you need on a technical level to build a reactive web application, it does not provide any functionality to style your web application. To that end, we rely on a library called MUI, formerly called Material UI when the previous group started their project. MUI provides components for React, with a component simply being a distinct element on a web-page; MUI provides everything from buttons to alerts to icons in the form of React components. These components are all based off of Google's Material Design language, which lays out a set of rules for designing good user interfaces. Additionally, MUI provides many utilities for laying out and styling pages, such as surfaces to place content on and grid layouts. Virtually everything you see in the staffing tool is a MUI component that has had custom styling and functionality placed on top of it.

Most of the libraries that our front-end depends on serve to facilitate the construction of the user interface, but we rely on several support libraries in other parts of our application as well. In particular, we use an authentication solution called Auth0 and its accompanying JavaScript library to authenticate users. This allows for users to seamlessly sign up from their web browser and for us to restrict access to resources. Every request that is sent from our front-end to our back-end has an accompanying

token generated by Auth0, which the back-end can verify. Additionally, various parts of the user interface are gated based on the user's Auth0 roles, which define what they should have access to: managing employees, managing projects, managing rates, and filling out timesheets. These roles can be mixed and matched as the administrators see fit, with a user having any combination of them. Above all else, however, Auth0 provides 7Factor with the peace of mind of knowing that their staffing tool is secured by a well-engineered and battle-tested piece of enterprise software.

User Interface Mockups

One of the major goals we had when we inherited this project was to improve the user interface, since the previous team focused more on setting up the back-end and database and then getting the user interface into a functional state. We made some mockups using Figma, a cloud based mockup tool, and decided on a design that we thought would both look good and easy to use

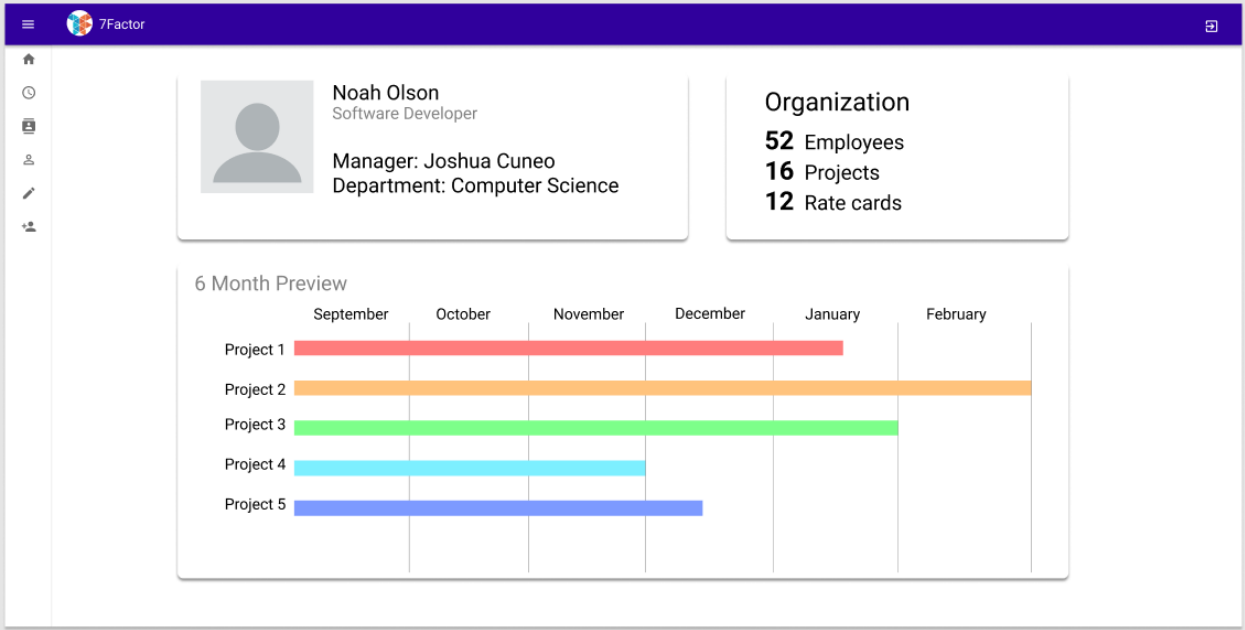


Figure 1: Mockup for the home screen

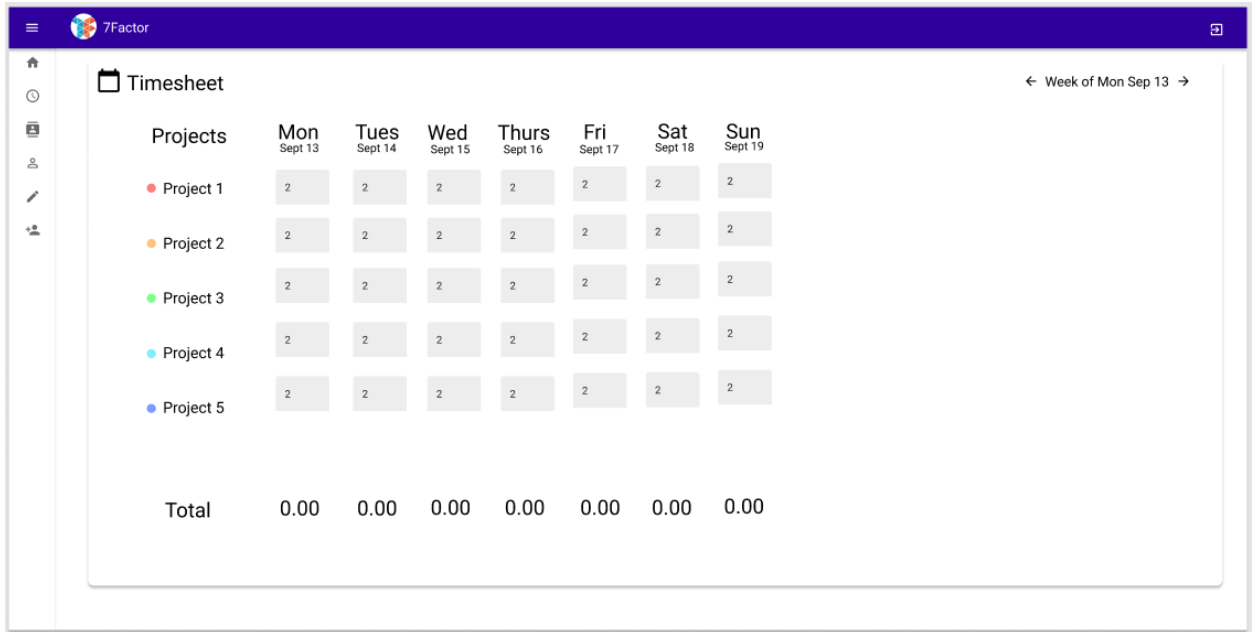


Figure 2: Mockup for timesheet screen



Figure 3: Mockup for employees screen

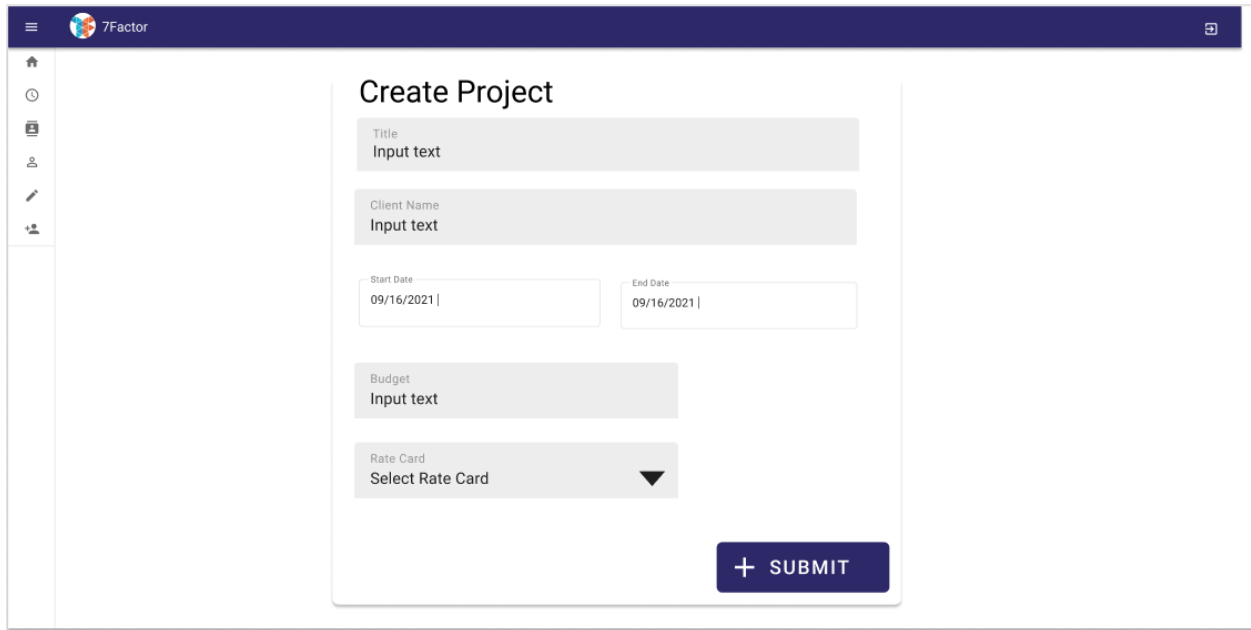
The image shows a web application interface for creating a project. At the top, there is a dark blue header with the '7Factor' logo on the left and a square icon on the right. Below the header is a vertical sidebar with icons for home, refresh, list, edit, and user. The main content area is titled 'Create Project' and contains several input fields: 'Title' (Input text), 'Client Name' (Input text), 'Start Date' (09/16/2021 |), 'End Date' (09/16/2021 |), 'Budget' (Input text), and 'Rate Card' (Select Rate Card with a dropdown arrow). A dark blue button with a white plus sign and the text '+ SUBMIT' is located at the bottom right of the form.

Figure 4: Mockup for dialog to create a project

We decided that most of the screens (e.g. employees, rate cards, projects, clients) would follow the card design we made in Figures (1) and (3). Figure (3) would be the main screen where you can see cards that represent the employees, rate cards, projects, and clients that already exist, and Figure (4) would be the dialog that shows up when you want to add or edit an item. We thought that keeping the same design throughout the software would allow users to easily understand the information presented to them and. Additionally, this method uses the same patterns for viewing, adding, and editing data across every page, which simplifies learning how to use the application.

Results

User Interface: Before & After

After reviewing the state of the UI we received the project in, we decided that our primary objective would be a massive overhaul of the UI to take advantage of the already established back-end and create a stable, usable application. Once we created our mock-ups, we redesigned all aspects of the previous team's user interface work, making adjustments to also support our new features. When redesigning the user interface, we attempted to make as much functionality as possible consistent between pages. The result is a card-based layout on most pages with consistently placed buttons to add, edit, and delete or archive objects.

It should also be noted that every screen that made an API call to the database had an alert come up saying it was successful even if it was unsuccessful. This was probably for demonstration purposes, as it was unstyled and simply using the browser's default functionality; We removed this alert as one of our first changes to the user interface.

Login Page

The previous team implemented a basic front-end with the goal of showcasing the back-end and database components. They implemented a basic UI for adding Rate Cards, Employees, and Projects. They also had a login page that asked for Auth0 authentication after you clicked login.



Figure 5: Previous team's login page

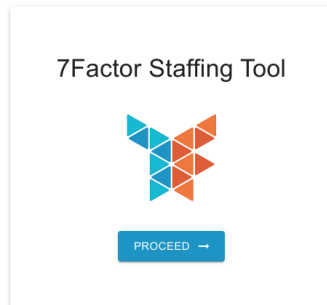


Figure 6: New login page

The new login page functions the same as the old one, but we added an elevated card and modified the button to match our button styling in the rest of the application. After getting Auth0 authentication, you were originally directed to the time entry screen shown in Figure (7), as there was no home screen.

Time Entry Page

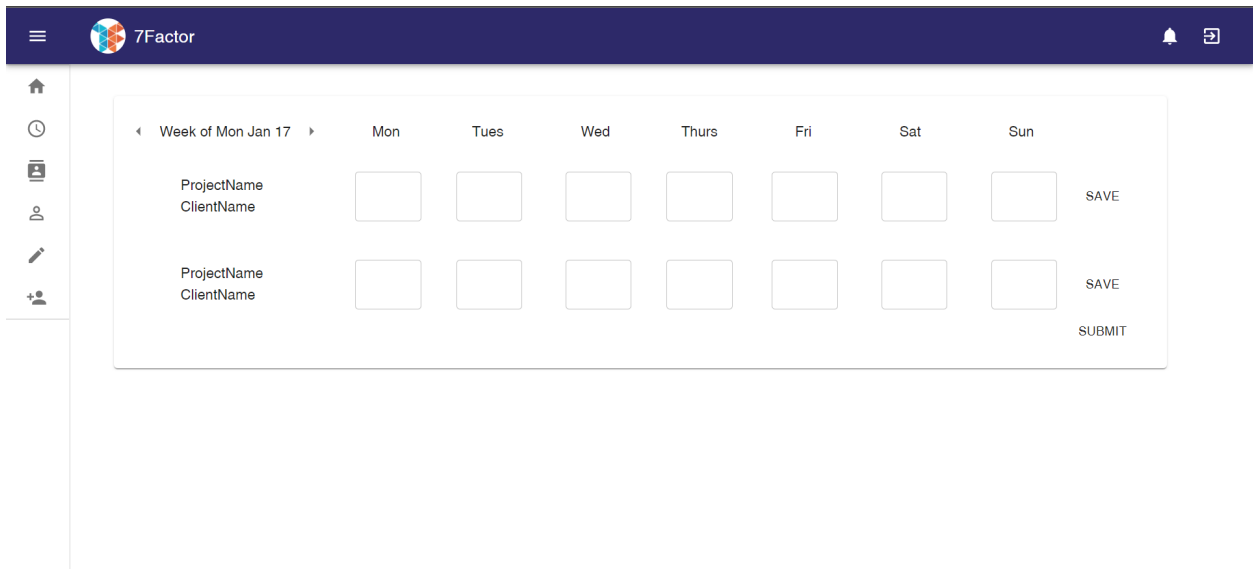


Figure 7: Previous team's time entry page

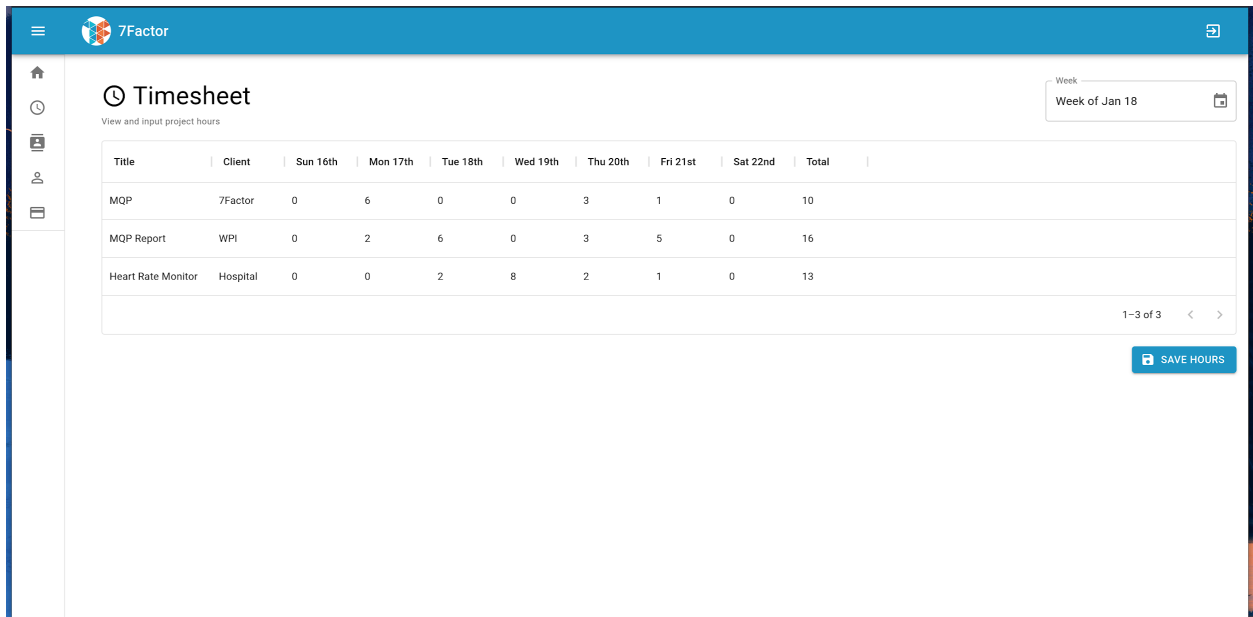


Figure 8: New time entry page

In Figure (7), two buttons can be seen in the top right. The first is a notification button (the bell) that did nothing when clicked on, and the other is a logout button which took you back to the login screen. The hamburger button in the top left allows for toggling between an expanded and minimized menu. Figures (7) and (8) show the

minimized menu. If expanded, the menu shows the titles of the pages to the right of the icons. This functionality was retained in our redesigned user interface.

The previous time entry screen in Figure (7) was purely for display as none of the buttons were functional, and the submit button just displayed an alert saying “Timesheet submitted” without doing anything else. This screen also did not adapt well to smaller screen sizes or zooming in. With our redesign of the timesheet seen in Figure (8), we opted to make the experience more user friendly, especially for navigating past weeks. Now there is a popup calendar for selecting dates, which will make navigation easier. The interface is also now fully functional, and the user can double click on any time entry in the table, enter a valid number of hours, and hit the save button to submit the changed time entries.

Rate Card Page

Next up is the rate card creation screen. This screen was functional; however it did not submit data properly to the database even though it displayed a success alert, so we had to quickly fix it so the created rate card was stored in the database successfully.

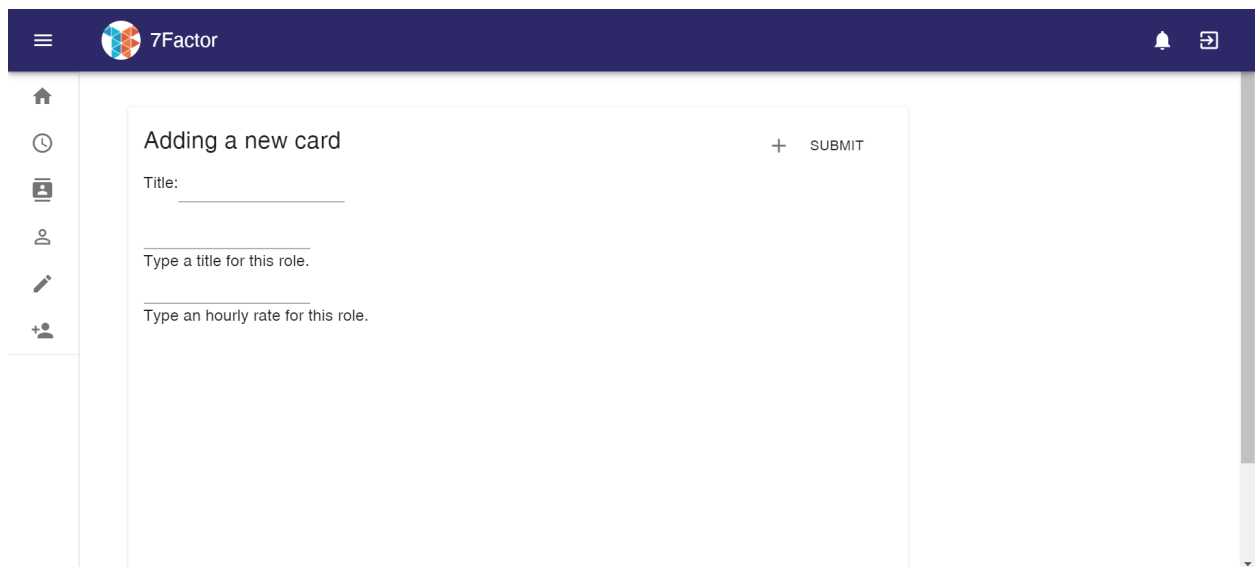


Figure 9: Previous team’s adding rate card screen with only one role

There were several other issues with this screen with the one being the title field was in a fixed spot on the user's screen. This caused it to overlap with the other entry fields when the user scrolled down as seen in Figure (11). Another issue was that roles could not be removed after they had been assigned.

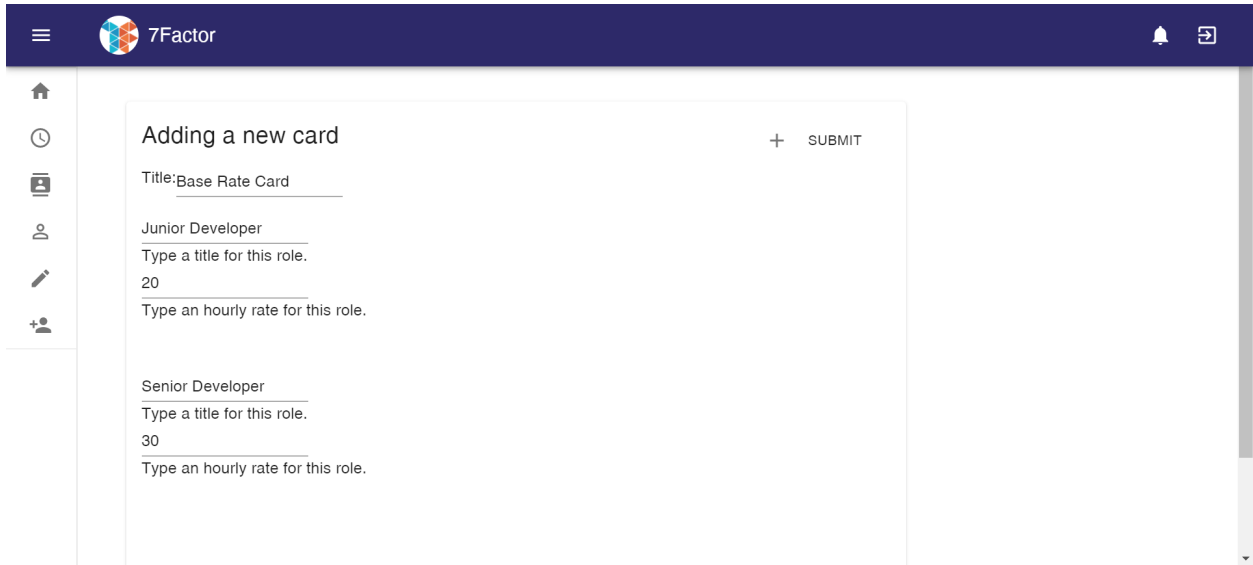


Figure 10: Previous team's adding rate card screen with multiple roles

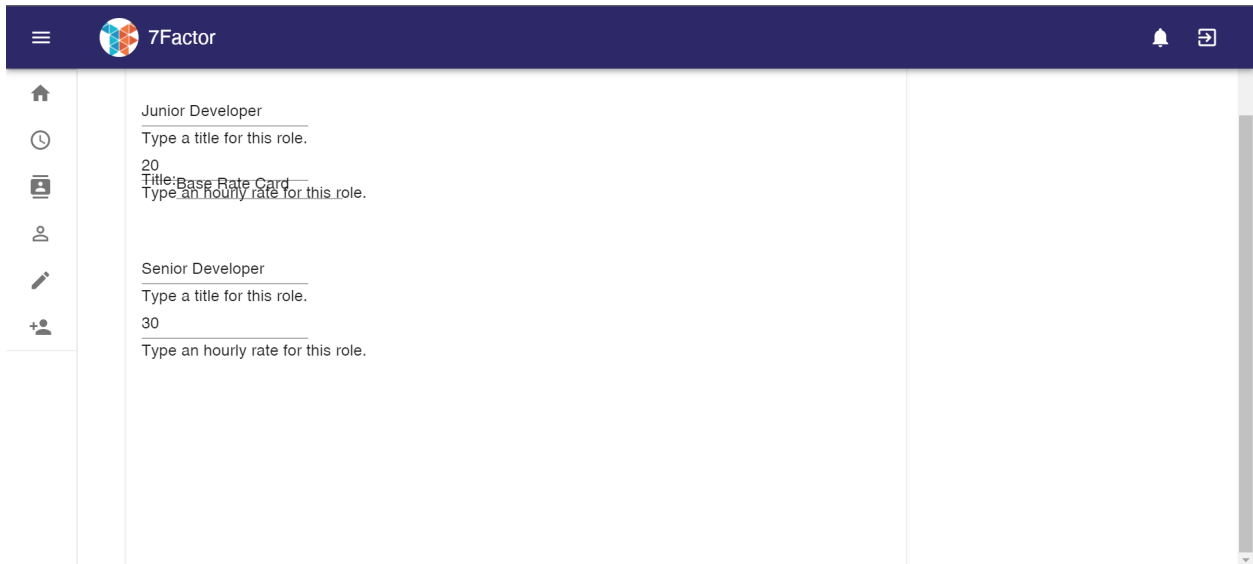


Figure 11: Previous team's adding rate card screen with title overlapping other fields

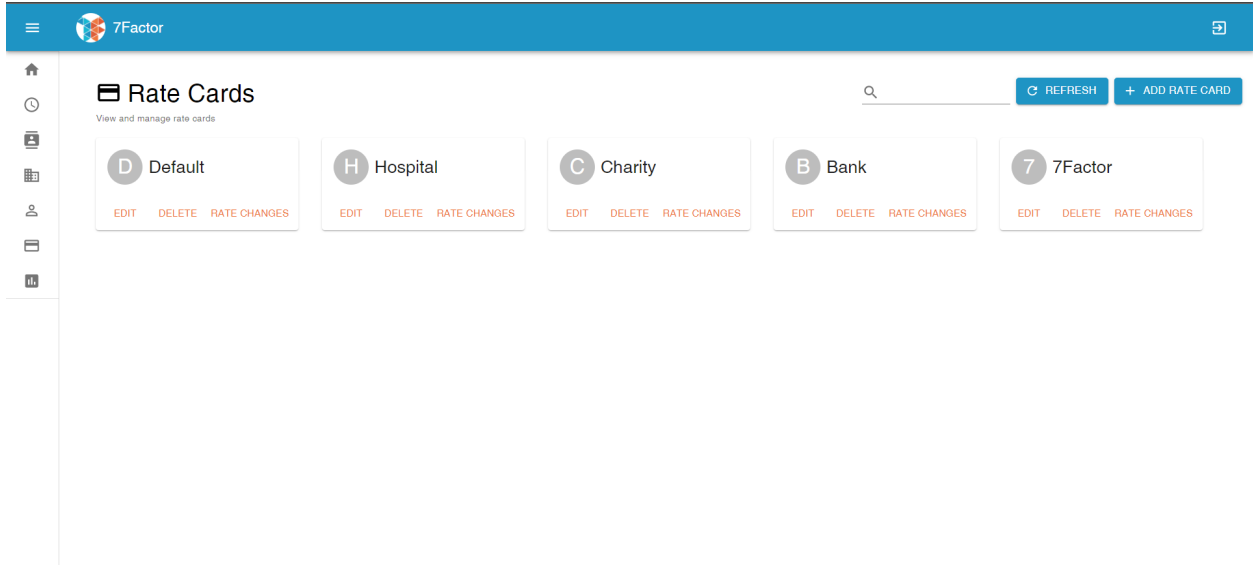


Figure 12: New rate card page

As with our other pages, the new rate card page follows a card-based layout where you can easily see every rate card. You can add rate cards using the button in the top right, and edit or delete specific rate cards by using the button on the card itself.

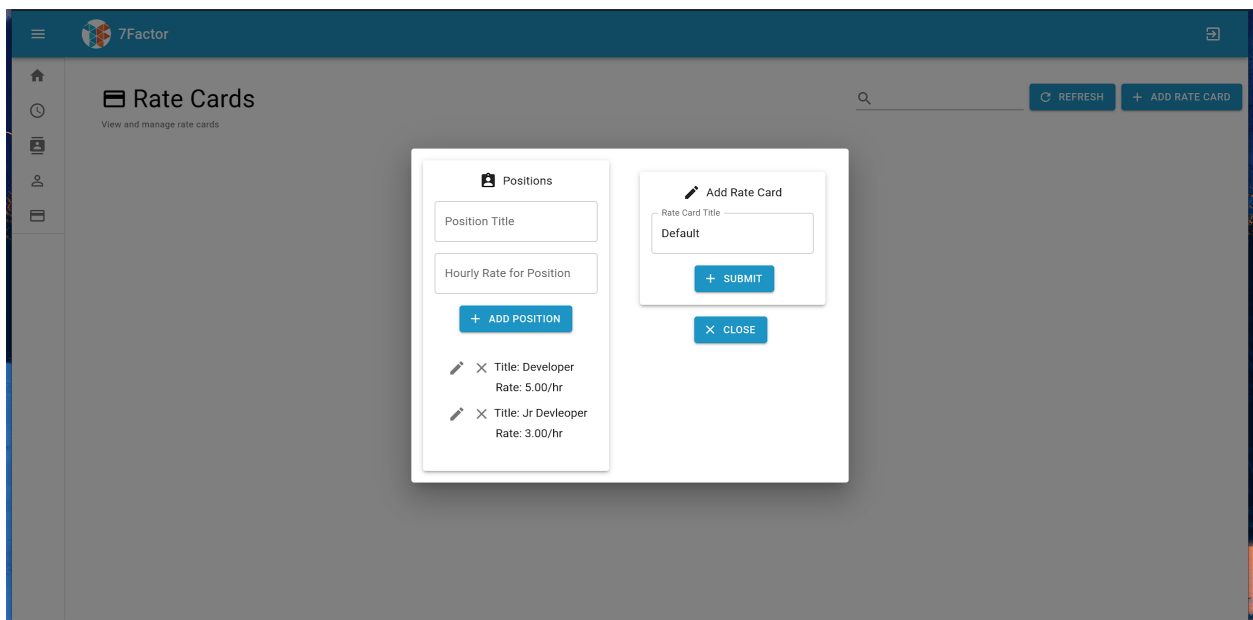


Figure 13: Each rate card can be edited and created using this dialog box.

We also added the ability to edit and delete positions on a rate card. You can now change the title for a position as well as the rate for it. When you change the rate, you also need to include a date where that rate change becomes effective as seen in Figure

(14). It is also possible to see the previous five rate changes for a position if you click the “Rate Changes” button on a rate card as seen in Figure (15).

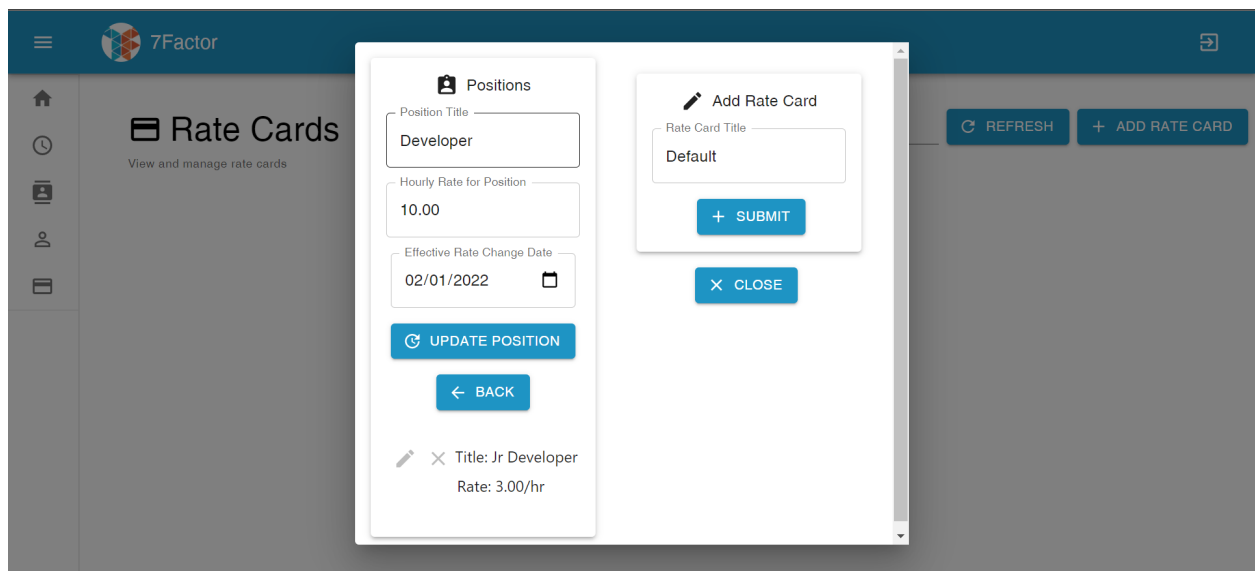


Figure 14: Editing a position on a rate card

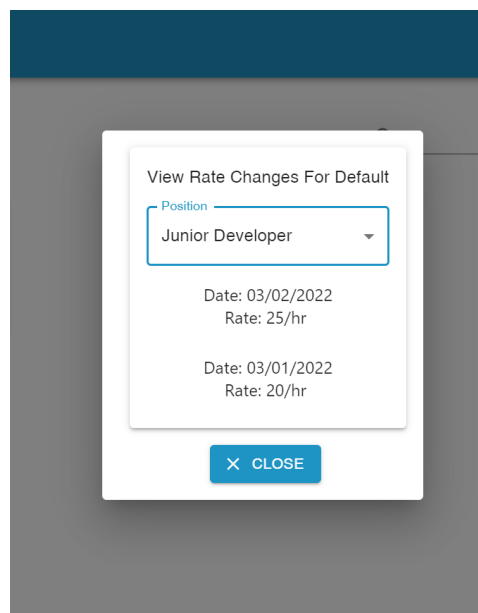
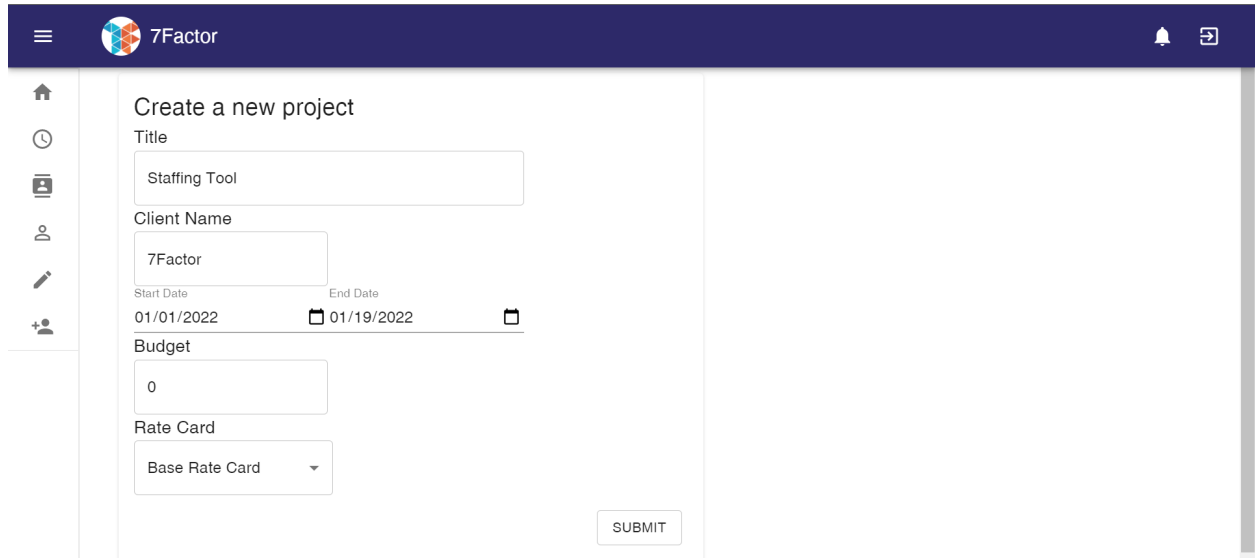


Figure 15: Viewing past rate changes for a position

Project Page

After creating a rate card in the old version of the web application, we could create a project on the create project screen.



The screenshot shows a web application interface for creating a new project. The header is dark blue with the 7Factor logo and navigation icons. A sidebar on the left contains icons for home, clock, calendar, person, edit, and group. The main content area is titled 'Create a new project' and contains the following fields:

- Title:** Text input field containing 'Staffing Tool'.
- Client Name:** Text input field containing '7Factor'.
- Start Date:** Date picker showing '01/01/2022'.
- End Date:** Date picker showing '01/19/2022'.
- Budget:** Text input field containing '0'.
- Rate Card:** Dropdown menu showing 'Base Rate Card'.
- SUBMIT:** A button at the bottom right of the form.

Figure 16: Previous team's project creation screen

This screen was fully functional except for the fact that it did not create a project if you selected a rate card to go with the project, so you had to leave that field blank.

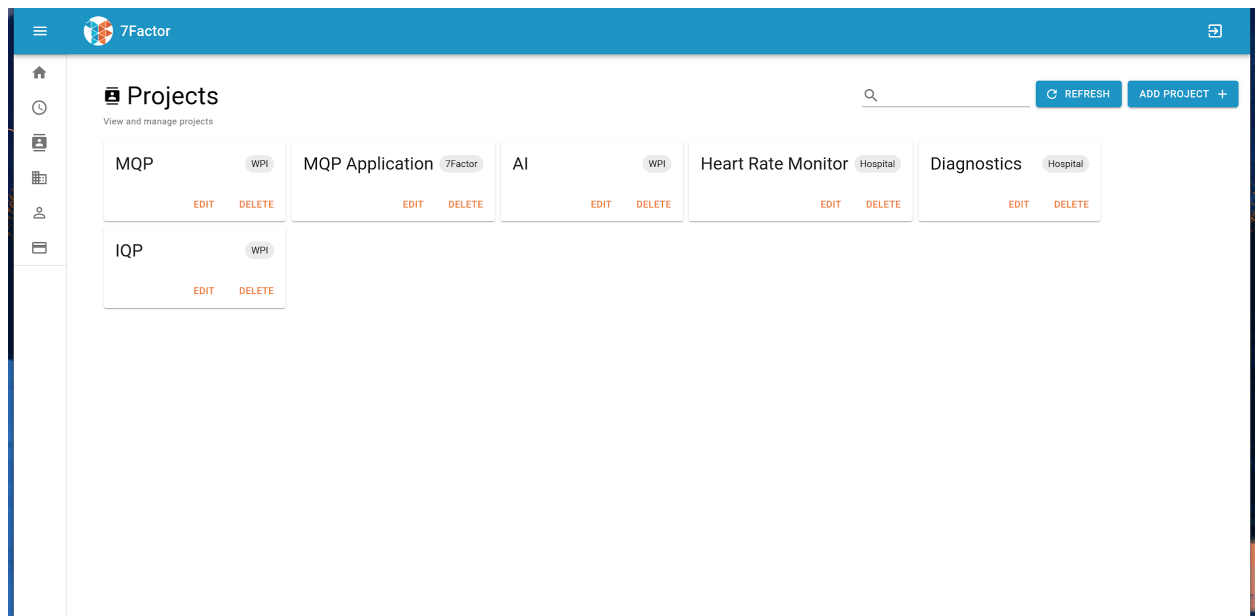
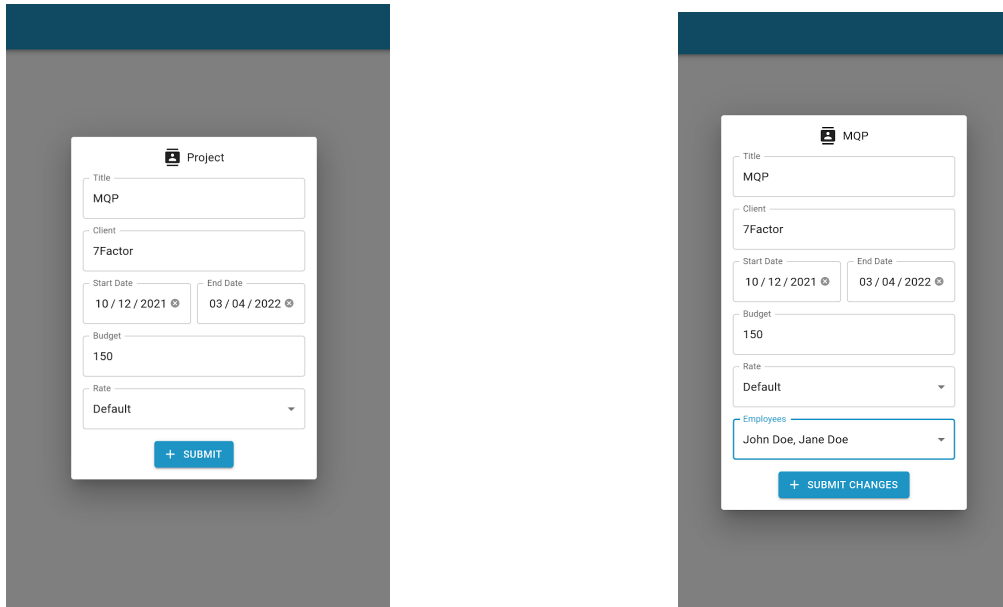


Figure 17: New project management screen

The redesign of the project screen also follows a card-based layout for viewing and managing projects. The user is able to search for projects by their name and client, and can add new projects using the button in the top right. Existing projects can be edited or deleted by using the buttons on the project cards.



Figures 18 and 19: Dialogs to Create and Edit Projects

One major change is the ability to assign employees to projects from the project management dialog, rather than doing this in the employee screen. This was included to allow managers to assign and unassign employees while making changes to the project. Additionally, this change allows for multiple employees to be assigned to a project in one edit to the project.

Employee Page

The second to last screen in the old version was the employee creation screen. There were no issues with this screen other than some overlapping elements with the rate card field that you can see in the figure below.

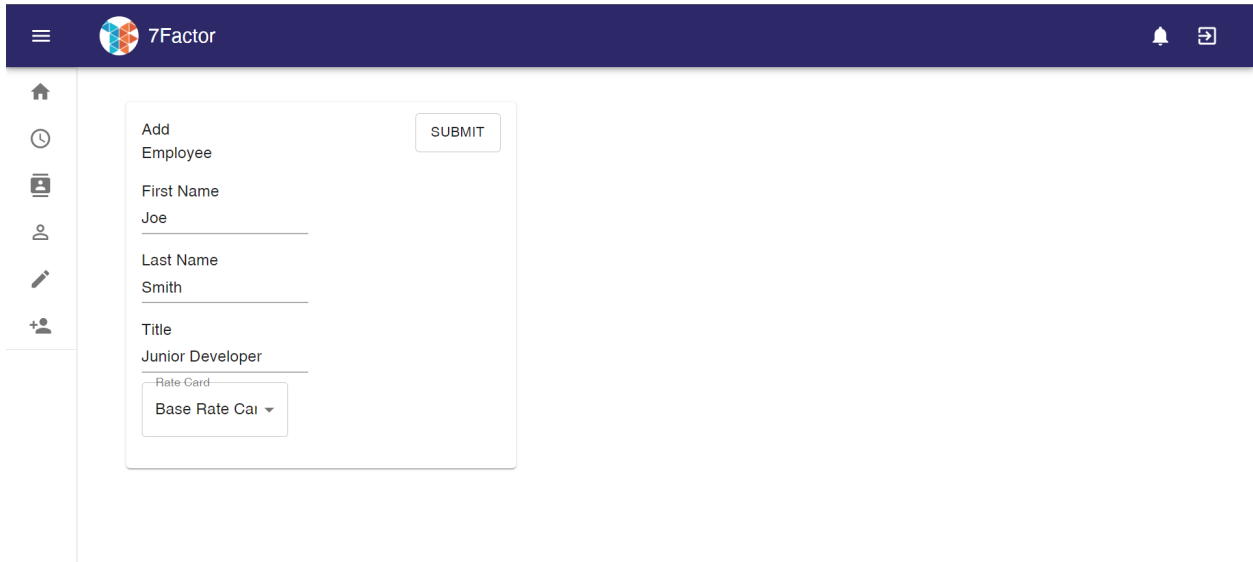


Figure 20: Previous team's employee creation screen

Finally, we have the people screen which initially crashed the site when clicked on. After fixing this issue we were presented with this screen. Please note that in our new version of the user interface this screen no longer exists, as we moved assigning employees to projects into the project editing dialog.

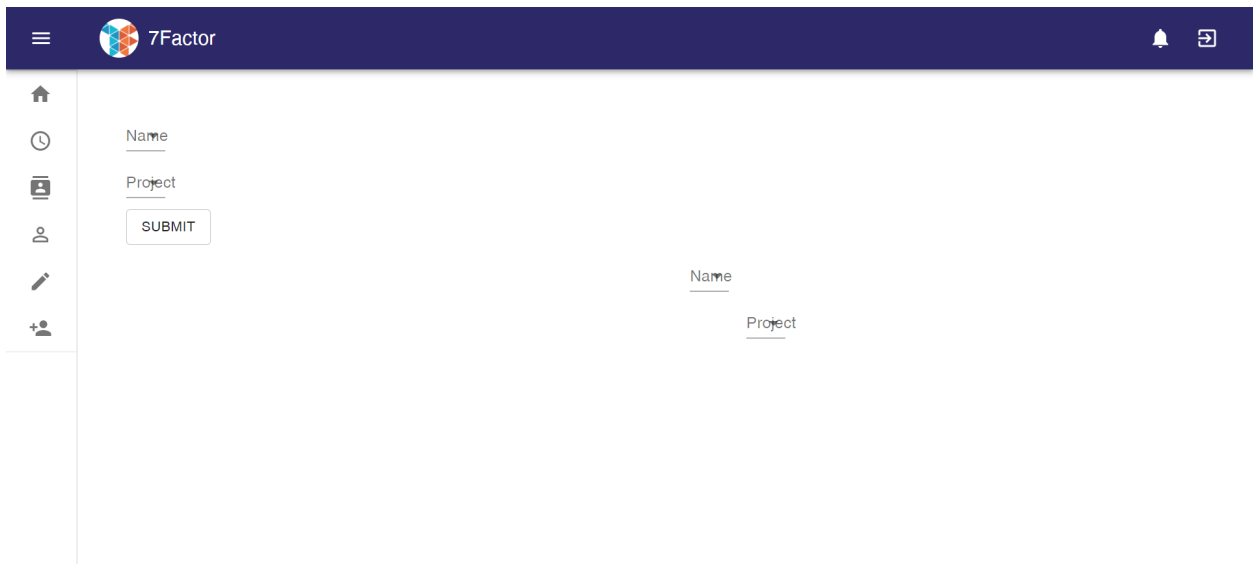


Figure 21: Previous team's people screen with no employees created

This page is very barren and has two identical pairs of drop-down menus. We believe the pair on the right was for demonstration purposes since there were generic

employees there that we did not create. This screen does look better once we have an employee in the database, as seen in the next figure.

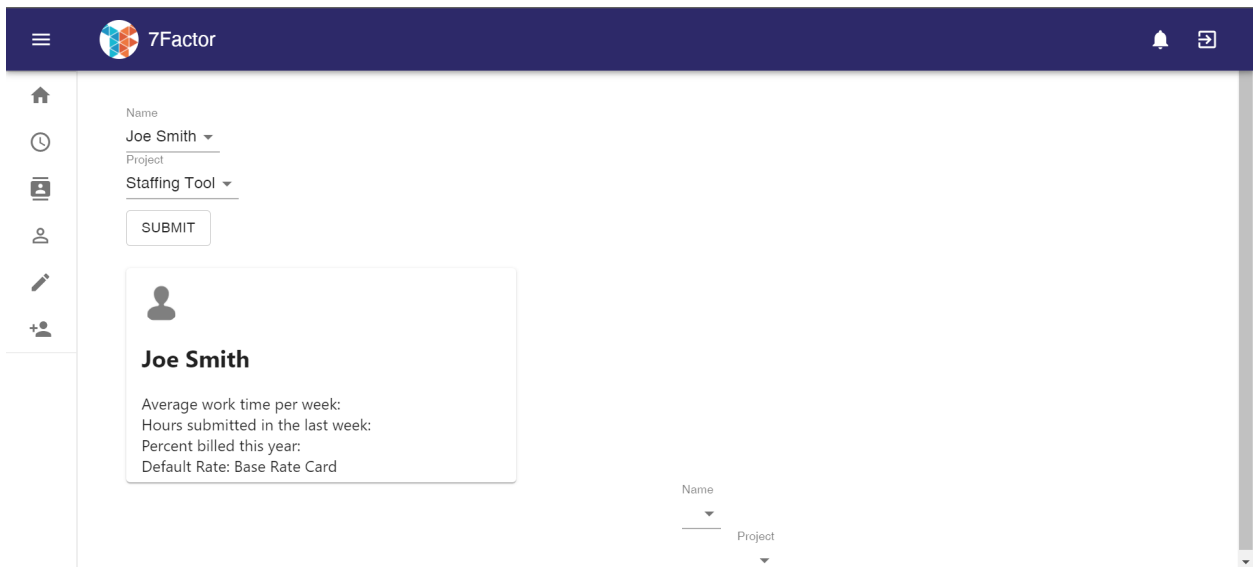


Figure 22: Previous team's people screen with employee present and fields filled

Now there is a sizable portion of the screen taken up by the new employee, along with some data about them that could be extrapolated once the application was complete. We can assign the employee to the project we created as well, but there is no way to see that on the front-end.

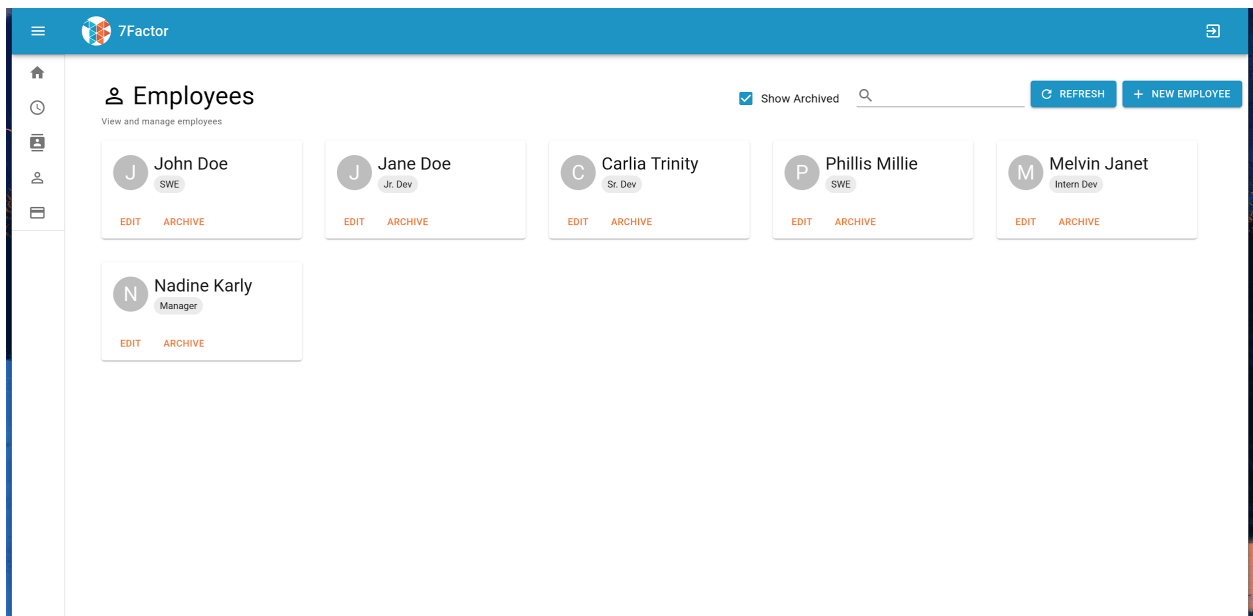
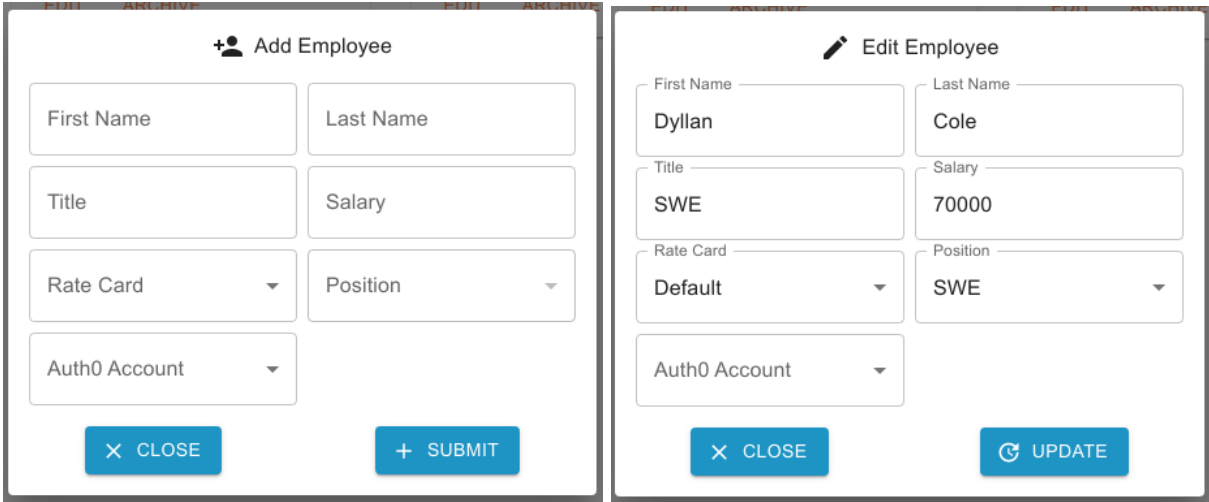


Figure 23: New employee page

In the same style as our other pages, we modified the employee page to use a card-based layout for viewing and managing employees. As with the rate card and project pages, you can click the button in the top right to add a new employee, bringing up the dialog shown in Figure (24), or click the edit button on a card to edit an existing employee, which brings up the dialog shown in Figure (25). A key difference is that employees can not be deleted, but only archived and unarchived. The show archived checkbox at the top of the screen in Figure (23) can be used to toggle whether or not archived employees are shown as part of the list.



Figures 24 and 25: Dialogs to Create and Edit Employees

User Interface: Additions

After modifying the old pages to improve their functionality, we added two new pages: a home screen, and an analytics page. Whereas the user was previously redirected to the time entry page after logging in, they are now sent to the home page.

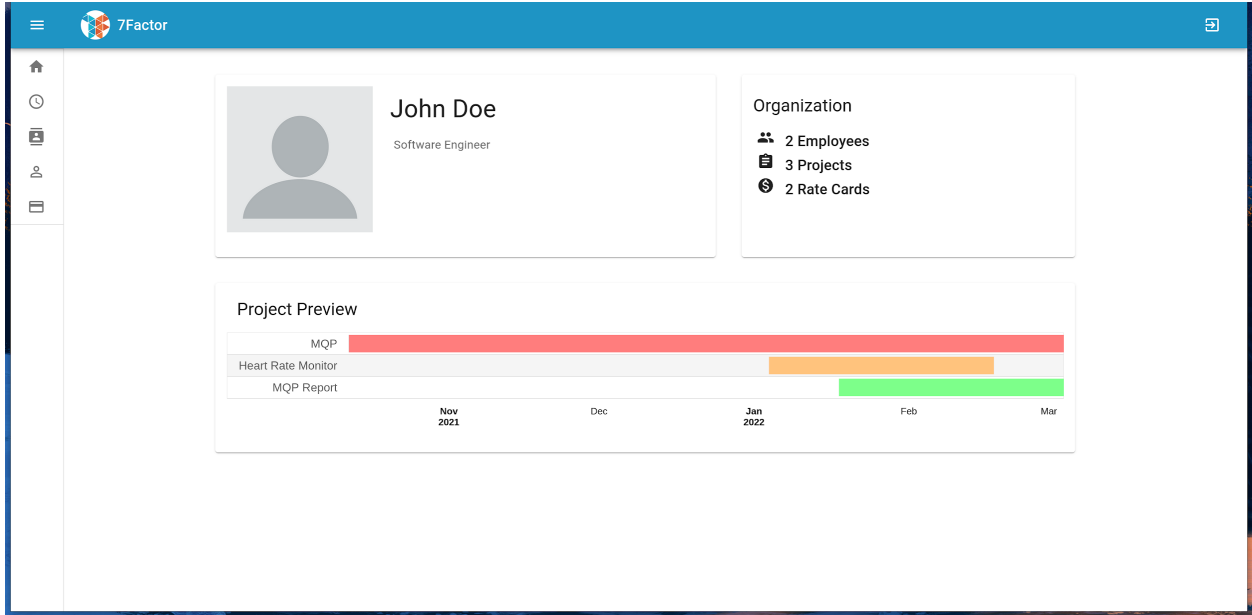


Figure 26: Home Screen displaying information about the logged-in user

The home screen seen in Figure (26) gives the user a brief overview of organization statistics and the timeline of any projects to which they are assigned. The project assignment information uses our newly added association between Auth0 users and employees.

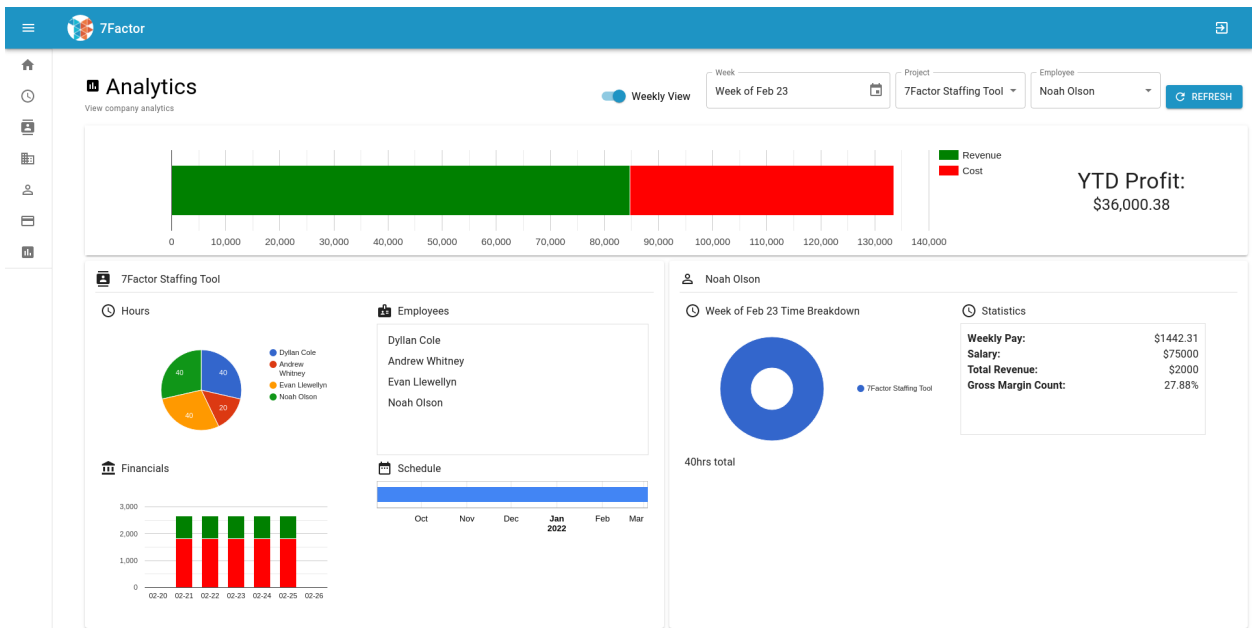


Figure 27: Weekly view of analytics page

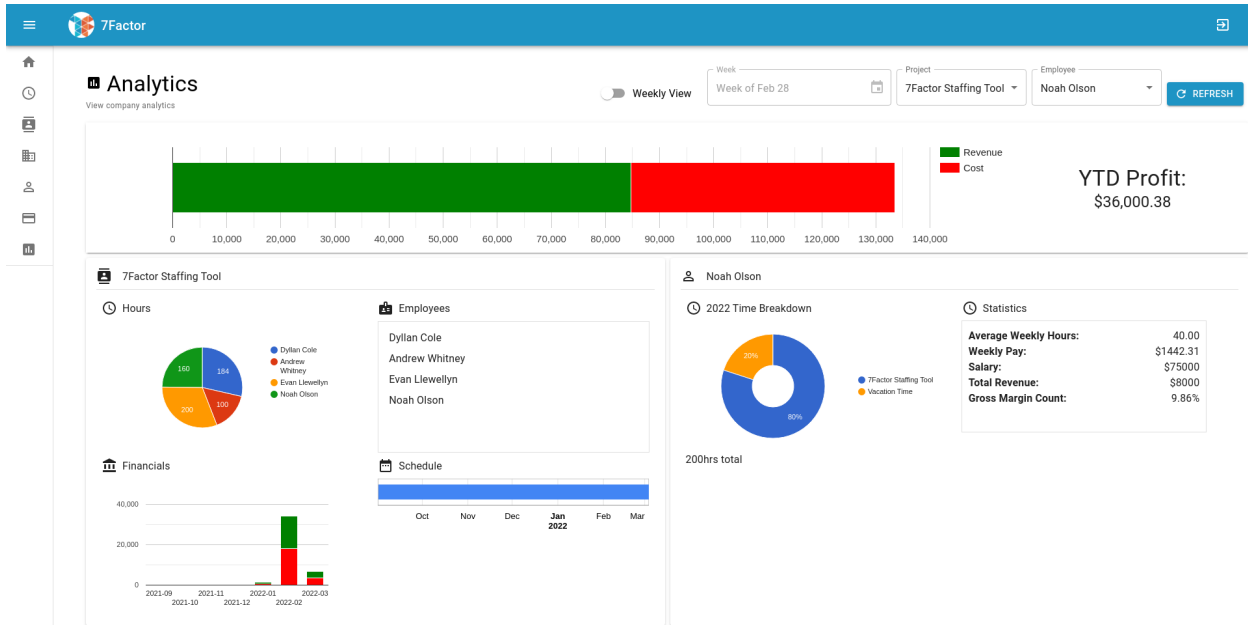


Figure 28: Overall view of analytics page

Our most substantial addition to 7Factor’s staffing tool was our analytics page. This provides metrics and data visualizations which can help management monitor its projects, employees, and broader financial position. Here we present a simple way to understand performance at a glance.

Project analytics, which can be seen in the lower left card in Figure (28), presents a per-project overview. It shows the schedule of a project, the employees assigned to a project, how many hours each employee has spent on the project, and a financial breakdown of the project’s cost and profit. Additional information can be seen by hovering over the elements: the schedule will show the specific dates and duration, the hours pie chart will show the actual hour counts, and the financials bar chart will show the dollar value of each bar. As can be seen in Figure (27), when in the weekly view the financials section will change to show a per-day breakdown.

The employee analytics component displays a project contribution breakdown for a week or year (depending on the weekly toggle) and general employee statistics. Most notably, managers can see the average weekly hours an employee works and their gross margin count, which is one common metric for profitability.

$$\text{Gross Margin Count} = \frac{\text{Revenue} - \text{Cost}}{\text{Revenue}}$$

Back-End Changes

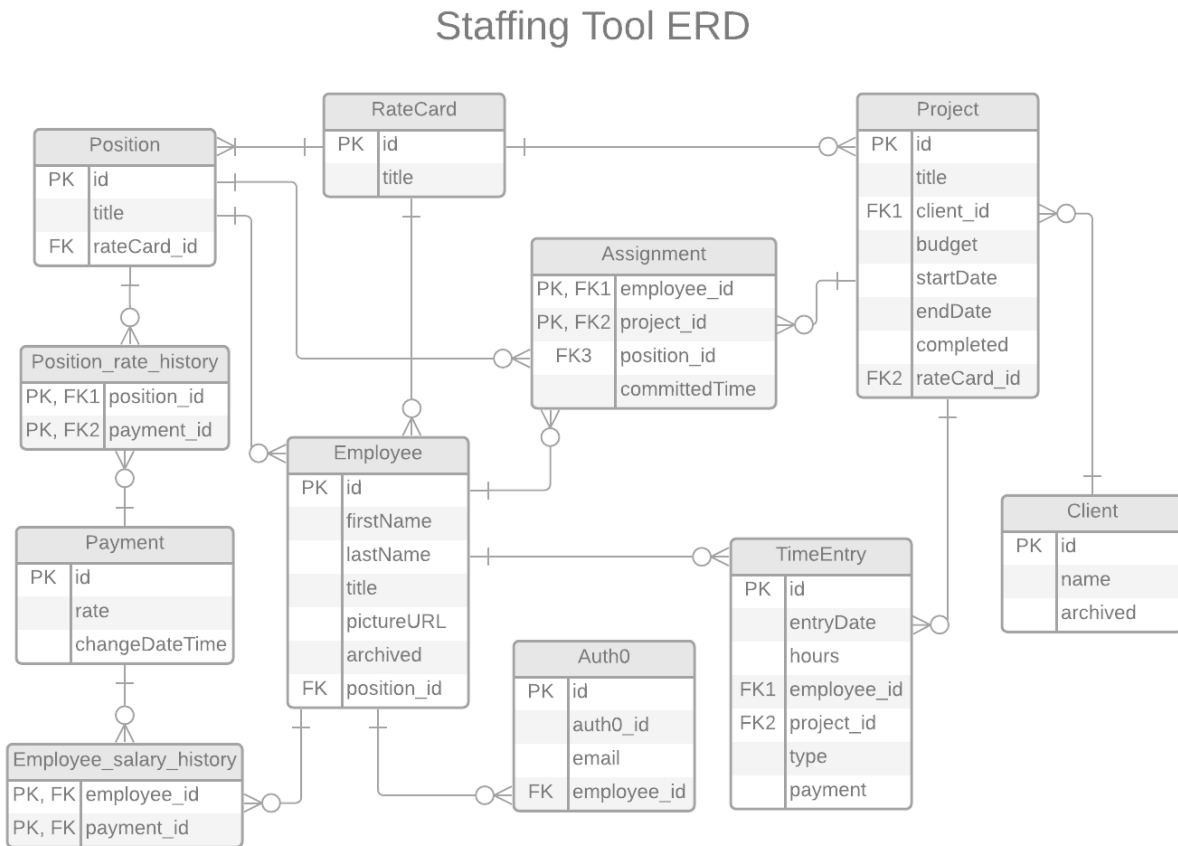


Figure 29: Staffing Tool Database Entity-Relationship Diagram

The back-end, while mainly complete, was expanded and modified to accommodate new features to the front-end. The most notable changes were to rates and time-keeping. When we began implementing time entry on the front-end, we looked at the back-end and saw a few fundamental issues preventing us from calculating default payments for a time entry:

- Employees had a default rate card which was just a string, not a rate card ID. It is unclear what this was intended to be used for.

- Employees had a role which was also just a string, unconnected to our concept of positions. It is also unclear what this was intended to be used for.
- Assignments had a custom rate history, but also had a position associated with its own rate history, which was obviously unnecessary.

To solve these problems, we decided on a simple scheme to determine what payment was appropriate for a given time entry. We first altered the employee object to remove the role and default rate card, replacing it with a rate card and a position on that rate card. With that change, we then refactored the payment calculation code so that the payment uses a position on the specific project assignment if it exists; otherwise, it uses the employee's position. This results in a default billing rate associated with an employee that can also be overridden on a per-project basis if required.

One major requirement which was not fully implemented last year was the use of Auth0 to secure the application and provide employees a way to login. The previous team began implementing Auth0, but it was not in a functional state. There was verification of Auth0 tokens sent from the front-end to the back-end, but the front-end was misconfigured in a way such that it could not generate proper tokens. We fixed that misconfiguration, and also changed the back-end so that Auth0 IDs can actually be verified and linked with employees in the database.

Role-Based Authentication

In order for this application to have a practical benefit, it was necessary to include a modular permission system. Since our application incorporates the ability to create, modify, and delete employees, projects, and rate cards, partitioning off who has access to these parts of the system is important.

Since the project already utilized Auth0, it was a logical step to associate roles and permissions with the Auth0 accounts. Roles are a built-in feature of Auth0, so the implementation for adding and removing roles in this application was simple.

Roles

+ Create Role

Create and manage Roles for your applications. Roles contain collections of Permissions and can be assigned to Users.

Name	Description	
employee-manager	Employee with access to the employee management screen	...
project-manager	Employee with access to the project management screen	...
rate-manager	Employee with access to the rate card management screen	...
standard-employee	Standard Employee has access to fill out their timesheet	...

Figure 30: Roles on Auth0

The screenshot shows a user profile for a user with initials 'AN'. The user's name and user ID are redacted with black boxes. Below the profile information, there are tabs for 'Details', 'Devices', 'History', 'Raw JSON', 'Authorized Applications', 'Permissions', and 'Roles'. The 'Roles' tab is selected. Below the tabs, it says 'All Roles assigned to this User.' and there is an 'Assign Roles' button. A table lists the roles assigned to the user:

Name	Description	Assignment	
analytics	User has access to view Financial, Employee, and Project Analytics	Direct	🗑️
client-manager	Employee with access to the client management screen	Direct	🗑️
employee-manager	Employee with access to the employee management screen	Direct	🗑️
project-manager	Employee with access to the project management screen	Direct	🗑️
rate-manager	Employee with access to the rate card management screen	Direct	🗑️
standard-employee	Standard Employee has access to fill out their timesheet	Direct	🗑️

Figure 31: User with Roles on Auth0

When a user does not have access to a page, the application will render a “No Permission” page instead of the page the user is trying to access. Also, when a user does not have permission to access a certain part of the application, that part will not be shown on the sidebar.

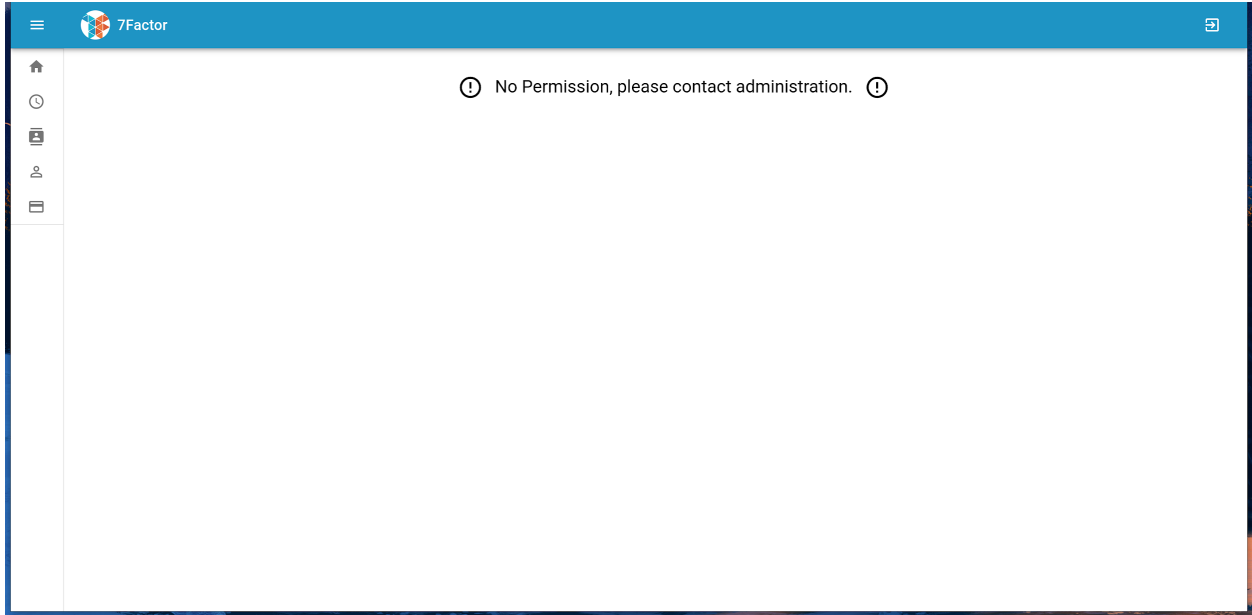


Figure 32: If a user does not have access to part of the application, this screen will be displayed.

The access system we currently have in place can be grown into a more complex permission system; however, it provides the fundamentals to allow modular access to the application, which will fit the needs of employees at 7Factor.

Recommendations and Future Work

The area of future work that we believe is most important is security. We already have a very secure authentication system thanks to our usage of Auth0, but we did not implement much server-side validation of requests. Every request sent to the server features a token generated by Auth0, and the server then checks this token before it processes the request, but this does not take our implementation of distinct roles into account. A user cannot easily access the parts of the application that they do not have roles for, but front-end Javascript code is easily modifiable; someone who was sufficiently motivated and had a valid Auth0 user could break into parts of the application that they do not have access to and send requests that the server would accept. Additionally, since the Auth0 tokens are stored on the client before being sent to the server, it would be entirely possible to simply intercept that token and then use it to send bogus requests.

Our lack of server-side role authentication is a security hole, but we would like to not overstate the threat. Any random person cannot gain access to the staffing tool over the internet. To abuse the role issue, a valid account is required, which would be obviously restricted to only 7Factor employees. Additionally, it would be trivial to identify which employee abused their staffing tool login to gain access to endpoints that they should not have had access to. That is why we did not feel it was a high priority issue during our work on this project.

While implementing additional functionality during our project, we always tried to follow best practices and write well-performing code. We did, however, inherit a design decision from the previous group that has a potential negative front-end performance impact: the direct server-side representation of objects is sent over to the front-end with each API request, including many relations between objects that often result in very large requests. For instance, for us to get a list of rate cards on the front-end, we request the rate card data from the back-end. The rate cards carry information on the back-end

describing positions, employees, and projects. Since we just directly serialize the rate card and send it to the front-end, that means that we also send along all the information on every employee, project, and position, but that is unnecessary if all we want to do is get a list of rate cards. We suggest that a good solution to this problem would be implementing Data Transfer Objects (DTOs), which are objects constructed specifically to hold only the information relevant to a given request. By implementing DTOs, one could drastically reduce the amount of information that is sent to the front-end in response to simple requests.

Responsive design was not communicated to us as an immediate feature that we needed to prioritize in our project, but in our many discussions with 7Factor we understood that responsive design would be a good feature to have down the road. Responsive design is a web development approach whereby you add functionality to a website to make it adapt to different screen resolutions and aspect ratios. Responsive design has come into the vogue with the rise of smartphones; before responsive design, companies might have to make an entirely different website that would be usable from a smartphone, but now the same website can be used across a phone, tablet, laptop, desktop, or even more niche devices such as smart fridges. The benefits of responsive design are clear; implementing responsive design principles makes a website usable on any device with far less work than it would take to entirely reimplement the user interface for every device.

What would responsive design entail in the context of our project? Thankfully, our UI component library MUI contains many features to make responsive design easier. Traditionally, one would manually write CSS that could apply styles based on different screen sizes. With larger websites, however, this quickly results in a lot of duplicated and confusing CSS styles. To solve that problem, MUI provides responsive grids and containers and a breakpoint API. The UI could be refactored to use said grids and containers, which would then automatically handle scaling and placing elements based on screen size. For any elements that cannot be switched over to use grids and containers, MUI's breakpoint API is essentially a version of CSS media queries that is

fine-tuned to be used from React. By implementing a few different breakpoint sizes, one can then use the breakpoint API as well as MUI's built-in support for the styled library to write responsive CSS directly in the JavaScript source code. This also has the benefit of eliminating a lot of the duplicated CSS code that would result from specifying traditional media queries directly in a CSS stylesheet.

Conclusion

The limitations and unnecessary complexity of TimeIQ caused 7Factor to seek out a new, more tailored staffing management tool. Our team picked up the work of our predecessors to create a tool which provides time entry sheets to employees and admin management of clients, employees, projects, and rate cards. Our platform additionally displays valuable metrics concerning 7Factor's finances, projects, and employees. All pages are styled in a consistent manner which is user-friendly and aesthetically pleasing. Lastly, we finished the Auth0 implementation started by the previous team and added role-based access control for more granular security.

Bibliography

7Factor Software DevOps and Cloud-Based Solutions. (n.d.). 7Factor Software DevOps and Cloud-Based Solutions. Retrieved December 13, 2021, from

<https://www.7factor.io>

Auth0: Secure access for everyone. But not just anyone. (n.d.). Auth0. Retrieved January 17, 2022, from <https://auth0.com/>

DB-Engines Ranking. (n.d.). DB-Engines. Retrieved January 17, 2022, from

<https://db-engines.com/en/ranking>

Group, P. G. D. (2022, January 17). *PostgreSQL*. PostgreSQL.

<https://www.postgresql.org/>

Manifesto for Agile Software Development. (2001). <https://agilemanifesto.org/>

Material Design. (n.d.). Material Design. Retrieved January 17, 2022, from

<https://material.io/design>

MUI: The React component library you always wanted. (n.d.). Retrieved December 13, 2021, from <https://mui.com/>

React – A JavaScript library for building user interfaces. (n.d.). Retrieved December 13,

2021, from <https://reactjs.org/>

Spring makes Java simple. (n.d.). Spring. Retrieved December 13, 2021, from

<https://spring.io/>

Time IQ. (n.d.). Retrieved December 13, 2021, from <http://www.timeiq.com>

Appendix

A. Development Environment Setup

These instructions assume that Linux is being used. You must install all of the following to run the staffing tool:

- Docker
- NPM
- Git

If you are using Ubuntu, you can run the following command in a terminal to install all of these packages at once:

```
sudo apt-get install nodejs npm docker git
```

Additionally, once Docker has been installed the Docker daemon must be running in the background. This may happen automatically, but if it does not you can run the following command in the terminal on most Linux distributions to start it:

```
sudo systemctl start docker
```

B. Running the Staffing Tool

Please Note

If you have any difficulty getting the staffing tool running, please feel free to email the authors of this report at their emails on the Digital WPI page for this project report. Our team had considerable difficulty initially running the staffing tool and we wish to spare any future groups from that fate.

Instructions

To run the staffing tool, you must first download the source code for all the separate components:

- Front-End (<https://github.com/7Factor/mqp-staffing-tool-ui>)
- Back-End (<https://github.com/7Factor/mqp-staffing-tool>)
- Database (<https://github.com/7Factor/mqp-staffing-tool-db>)

For each of these, use the `git clone [URL]` command, keeping in mind that you must have first been given access to the github repositories by 7Factor. Once you have cloned all of the repositories locally, you should first start up the database, then the back-end, then the front-end. In a terminal opened in the database folder then the back-end folder, run:

```
./env/_up.sh
```

Finally, in the front-end folder you must run the command:

```
cd src && npm install && npm run start
```

Once you have run the command to start the front-end, your browser should open to a development build of the front-end. If the database and back-end have been set up properly, you should then be able to interact with the application normally. Please keep in mind that you may not be able to access certain parts if your Auth0 user does not have the corresponding roles assigned to it, however. These roles can be assigned in the Users section of Auth0. All of these instructions are also present in the readme files of the individual repositories.