Capstone Seminar: SitesMobile Final Report

Cassie Beisheim, Karch Hertelendy, Katie Jackson, Michael Oreto, and Tristan Winship

Department of Engineering and Information of Technology, University of Missouri

INFOTC 4970W: Senior Capstone Seminar

Professor Gillian Maurer

May 10, 2024

Abstract

Over the course of 14 weeks, the SitesMobile team embarked on a project to update the tracking system utilized by Mizzou's Computing Sites. Our mission was to replace the reliance on Microsoft Forms and Excel Workbooks with a mobile application tailored to streamline the management of student employee tasks and computer lab statuses. This application, targeted at roaming student employees, facilitates essential tasks such as inventory counts, computer cleaning logs, and weekly form submissions. Moreover, it equips managers with the tools to access collected data and review detailed reports for progress tracking, evaluations, and site data. Our project timeline was thoroughly planned, comprising 4 weeks of conceptualization, 6 weeks of development, and 4 weeks of testing and implementation. Each team member assumed responsibility for specific feature development, fostering collaboration for user interface (UI) and backend tasks as required.

Key objectives included establishing a unified system for tracking employee activities and lab statuses, integrating a dynamic remote database (Firestore) for efficient data storage, and researching viable database options within budgetary constraints. Throughout the semester, we successfully implemented the core features of our minimum viable product. Users can now authenticate themselves, utilize live location and map functionalities, submit productivity forms (such as supply counts and computer cleanings), create, assign, and resolve issues, view lists of buildings and sites, and access submitted reports. Additionally, the application allows users to add and remove images, enhancing the overall user experience and data management capabilities.

Project Proposal

The project proposal for SitesMobile outlined a comprehensive plan to replace the current system of a multitude of Excel sheets and forms used by employees at the University of Missouri's Computing Sites department. The goal was to create a centralized hub and unified mobile solution for managing the 30 computing lab sites across campus, tracking inventory, and collecting data on computers cleaned by employees.

The proposal detailed the key functions to be developed for the app. These functions included an employee login, which is made sure to be a university email, a distinction on whether someone is an employee or a manager, creating an interactable map of the Computing Sites, creating forms for inventory counting and submission, and the ability to log cleaned computers at each location. The user profile page was envisioned to display personalized information such as assigned key sets, employee ID, email, name, and designated position.

To achieve seamless data management, the proposal called for using Firebase, Google's backend application development platform. This would allow the team to integrate Google's Firestore database, utilizing its NoSQL JSON format and real-time synchronization. Enabling efficient data submission and retrieval for inventory tracking and supply management across the different sites.

The project proposal defined a planned timeline, predicted obstacles, software employed, a timeline, and training needed.The outlined risk factors and obstacles such as the team's limited experience with full-stack development, the learning curve associated with Firestore integration, and the challenges of collaborating effectively using GitHub. Hardware compatibility considerations included ensuring the app's functionality on various iOS devices and the need for the development team to have access to macOS devices running Xcode. The proposed timeline divided the project into multiple phases, including conceptualization, which would last three

weeks, database development, one week, research and development, five weeks, testing and implementation, three weeks, UI polishing, one week, and documentation and presentation preparation, one week. In terms of deployment, the proposal outlined a strategy involving Karch, Katie, and Michael focusing on backend development and database integration, while Cassie and Tristan worked on frontend development and UI design. The team met twice weekly to collaborate and assist each other as needed. The testing plan entailed creating previews for each app view, conducting user testing with Computing Sites employees, and gathering feedback on usability, functionality, and overall user experience. The team also recognized the need to monitor the usage of Firebase's free tier during testing and development.

Methods

Following the team's initial research, we had created a timeline for our Project Proposal described above that we had made as accurate as possible, but expected to keep flexible. Going into Phase 2: Design and Discovery, we had already developed a good base for the app that installed the required packages (such as the Firestore and Google sign in SDK) and was connected to the Firebase console we created for the project. We had expected to create our user profile documents in the database, indicate user admin access, and implement the login/signup function of our app within the first week as well as create a list of Computing Sites dynamically loaded from the remote database. We were able to achieve these goals around 3 days after the target day, but the more complex application of the features were not as linearly developed as we had originally planned. We were able to log into our university Single-Sign-On accounts through the use of Google's SDK and create a new user login for our app, however, differentiating any Google account versus a University of Missouri account and preventing any non-Computing Site employees from accessing our database was yet to be established, as it was brought to our

attention at a later time. The computing sites were being read, but the views to display them weren't yet in their final form and were relatively rudimentary with some values still displaying as an ID instead of the intended name value (see Figure A1). These details would be addressed later in development as we diversified our work load. The team had learned later that our app development for each feature was not often linear in the beginning stages as we were focusing on proving we could access the data our proposal outlined first before moving on to polishing.

The next steps were to enter some starting data into the database in order to make sure we were pulling the data correctly. This was to be done manually through the Firebase Console and eventually was aided through the use of a Python program run in Visual Studio Code which would programmatically add in large lists of data such as in the Computers collection (see Figure A2, Figure A3). In conjunction with the data entry, the other half of the team was working on development of the MapView. Research on how to implement the MapView had started immediately at the beginning of the project, due to the team's inexperience with the limitations and capability of Apple Maps. The base function of loading a map, accessing the phone's current location, and displaying pins for each building, was implemented even faster than we had initially anticipated. Filtering the buildings by group and getting the coordinates dynamically were added soon after. Due to the fact that the map was displaying just buildings instead of computing sites or inventory sites separately, we had to create an additional view we hadn't anticipated in our original wireframe (see Figure A4). It was a transitory view meant for each building to list the computing and inventory sites that are within the building and redirect the user to the desired detail view.

The next goal was to create the DetailedSiteView which would display information such as basic site information, equipment, and poster images. This was created on time, but would still

need further expansion to display a list of computers, printers, and link to the map (see Figure A5). The cell view would also receive a redesign to integrate specific site pictures, campus group location (ie. G1, G2, G3, R1, and R2), site type, and icons to display the contents of each site at a later time. Shortly after the cell view was redesigned, a calendar section was added to the DetailedSiteView. The Division of IT currently uses the software 25Live to manage the academic and event scheduling around campus, including for some of the classroom sites (CollegeNET 25Live, n.d.). On the Division of IT's website, where it lists the computing labs available around campus, one can open a classroom site and access an embed of a simplified calendar for the current day (University of Missouri Division of Information Technology, n.d.). To print the calendar, the site opens in a new tab that provides a URL which contains the search query for the specific site as well as the date. By customizing the date in the URL, we were able to implement a date picker in the DetailedSiteView to open the event calendar for the specified date with the option to further open the link in the local device's browser (see Figure B1). This feature will be especially useful for roaming employees who often have to go into classrooms to perform duties, but want to be able to quickly check the classroom schedule on mobile before relocating.

Around the same time as creating the DetailedSiteView, we had planned to integrate the images for specific sites as well as posters. This task wasn't assigned until 9 days after the original target date for completion. This was due to the fact that we had to add in a few tasks that pertained to security first. At the time of making our proposal, we did not factor in time for writing and testing security rules (see Figure B2). After meeting with our advisor, we had prioritized this concern and moved onto images next. Images were added to the DetailedSiteView only one week after it was assigned.

As per the project proposal, the next checkpoint was to create the submission forms for employees to fill out such as Inventory, Hourly Cleaning, Site Captains, and Site Readys. The inventory submission was anticipated to be the most intricate and was assigned first. Due to the base app (app with login functionality and connection to Firebase) taking longer than expected in the beginning of the process, we were now around 2 weeks behind the original timeline. Nonetheless, the team was able to start on the Inventory submission less than a week after the completion goal. The sequence of views for the inventory submission was the result of a brainstorming meeting with the Team Lead at Computing Sites, Jessica Findlay. On top of creating the submission, it was a challenge to create the initial template for how the team would display form entry data throughout the app. Displaying not only the basic information like user and date of submission, the app would have to query a separate collection in the database to access the actual supply data submitted for that specific inventory entry (see Figure B3). As was predicted, this feature was quite complex and required around 18 days of the schedule to complete. Once complete, it was ready for implementation and would not need any more final touches in terms of user design (see Figure B4). However, originally, in order to control the views programmatically and move the user automatically back to the base view automatically once the form was submitted, a NavigationStack was used. During Phase 3: Implementation and Testing, it would later be changed to use NavigationLinks instead in order to be accessed by the building transitory view mentioned previously.

The Site Captain and Site Ready forms were being designed around the same time. The Site Captain form had originally only collected string values for things such as issues and labels, and at the time only allowed for one issue and one supply request to be submitted at a time (see Figure A6). Deviating from the original proposal, we ended up creating additional data

collections in Firestore and new data structures in the app in order to collect the issues and supply requests as data we could easily display and manipulate. That addition meant we would have to create two extra views as well in order to sort through the Issues and SupplyRequests. Again, this required us to add more development during the end of Phase 3 and meant that the Site Captain and Site Ready forms were not completely finished until late April.

The Hourly Cleaning form was completed around 12 days after the inventory submission form. The form itself was relatively simple, though required some more time to adjust the user experience (UI) design and create the summary view for the admins to access the submitted data. All four of these forms also took longer than expected, because as we developed it occurred to the team, rather than developing the admin panel and views all at once after all of the forms were complete, it was more beneficial to develop the submission form and its corresponding summary view at the same time. When doing this, it ensures the data that is being written will be able to be decoded in the intended format consistently with minimal code conflicts. As a form was being developed, its data structure that was being pulled from Firestore and decoded into the manager file may be changed in terms of variable name and data types; this could create unnecessary conflicts if the summary view was being developed by a different team member (see Figure A7).

As the form summaries were being created, the team also decided to implement more advanced searching and sorting features for the convenience of admin users. Not only did we want it in the lists of buildings and sites, we wanted to add it to computers, printers, and summary views. This took additional time in Phase 3 of our timeline, though the team determined it was worth the time.

Originally in the planning process, the team had expected admin users to use the Firebase Console in order to authenticate user emails and grant admin access. The process of doing so is

not too difficult, it is simply to create a document in each collection with the user's email or user id respectively, but requires knowledge of the console and precision in the labels one uses when creating the documents. We were able to fit in time to create a view for admin's to see a list of all users and filter by which users are authorized and which user profiles no longer have logins associated with them (in the case that a user deleted their account). That list would provide redirection into the AdminUserProfileView which would allow for the admin to add/remove user positions, add/remove the user from the Authorized Emails collection, add/remove the user from the Admin collection, and delete a user's profile from the database (see Figure B5). While improving that functionality, this was also around the time the team was able to add some more security through the sign in process, by adding checks in the code to prevent any non-umsystem emails from signing in, thus creating an account, and accessing the rest of the app.

Working beyond the form summaries and user accounts, we had decided to alter one of the features outlined in the proposal. Instead of generating a list of supply moves for an admin to generate, we decided to implement assigning and resolving issues. In order to do so, we had to create an extra view to view the detailed information of an issue and provide a toggle for resolving/un-resolving an issue as well as a drop down list of existing users. Upon selection of a user, the admin would be prompted with a button to assign the new user to the issue. The users would then need a way to see which issues were assigned to them, so we decided to create another additional view called Productivity (named Tasks in the TabBar). This would not only allow users to see the issues assigned to them (filtered by resolution status), but also which computing sites were assigned to them (called Site Captains) as well as recent Hourly Cleaning entries they've submitted.

One of the final views the team worked on was the KeysView. Since users had the ability to see their own key sets, we felt it was a logical step and within our reach to create an admin view that would be able to see a summary of which key sets had which keys and belonged to which user. Creating this view and implementing searching and sorting took around 2 days.

At this point in development, the project was approaching the end of the timeline. The final steps for the team were to update the login page (see Figure B6), implement a few minor changes from user feedback, add any final adjustments to streamline the user interface design, and to review the app for any bugs, redundancies, or vestigial features. We also ensured that we added alerts wherever relevant to not only warn the user about destructive actions, but also prevent employees from submitting incomplete forms. This process had taken around the last 3 weeks of our project. For the Issues and Supply Requests view, we added the ability to multi-select from the summary view and delete submissions in bulk. This not only was helpful in cleaning up the database from old submission documents created during development and testing, but would also be highly useful for admins in order to reduce clutter of temporary tasks assigned to employees. As for UI design, one change includes the sections in the AdminView, DetailedSiteView, and DetailedInventorySiteView being updated to appear more distinct and identifiable as well as less distracting in terms of size (see Figure B7). Some others were updating the MapView's campus group filter to display information more efficiently and appear next to the title on the screen, keeping it apart from the actual map contents (see Figure B8), and updating the original Site Ready form to use screen space more efficiently (see Figure A8). Following those adjustments the team had moved on to final documentation for the project.

Division of Work

At the beginning phases of the project, the team embarked on foundational activities, including brainstorming sessions, instructional tutorials, and tool research to align with our project goals and learn necessary technologies. However, as the project unfolded, each team member assumed distinct roles and responsibilities, capitalizing on individual expertise and preferences. Katie undertook the responsibility of Project Coordinator, orchestrating documentation management, overseeing the initial app setup via Xcode and GitHub, and leading team meetings to ensure cohesion and progress in the project. Katie also contributed significantly to Firestore and SwiftUI development, alongside the initial mockup of the application user interface. She was a critical member to the team's success for the delivery of the application due to knowledge of Computing Sites, Firebase, and SwiftUI development.

Cassie fulfilled the crucial role of Point of Contact, acting as the liaison between our team and external collaborators such as the learning support team and our advisor. Apart from communication responsibilities, Cassie played a pivotal role in SwiftUI development, particularly focusing on the MapView functionality and graphic design elements such as the application icon and login page. Michael handled many Swift UI views and their functionality, including inventory views and detailed site views. One crucial view being the SiteCaptainView, which submits various data entries to Firestore and admin users to review.

Both Karch and Michael took charge of Firestore data entry and security/authorization rules, ensuring the efficient and secure management of data within the application. Karch also handled admin user features such as adding images to sites, updating employee positions and user authentication. Tristan concentrated on SwiftUI development, particularly focusing on implementing the MapView and polishing SitesView functionalities within the application. Tristan also worked on the site ready form to handle submission of data entries similarly to the

site captain form. Overall, every member of the team contributed to the development of the project in a meaningful way, utilizing individual strengths for specific tasks.

Our team's division of work was instrumental in ensuring efficient progress and problem-solving throughout the project. With distinct areas of emphasis assigned to each team member, we utilized individual strengths and expertise to tackle specific issues and tasks effectively. This structured approach facilitated smoother coordination during our bi-weekly meetings and provided clarity on who to approach for assistance in specific development areas.

Our two weekly meetings also served as crucial touchpoints for project updates, issue assignment, and collaborative problem-solving. Consistent communication during these meetings promoted a deeper understanding of the app's functionality and ensured alignment with project goals. In addition to regular meetings, we utilized a variety of platforms to facilitate effective collaboration. GitHub served as a central hub for project management, allowing us to manage version history, create and merge branches, and reference each other's work efficiently. Platforms like iMessage and Discord provided convenient channels for real-time communication, enabling us to share updates and seek assistance as needed, fostering a supportive team environment.

Furthermore, Google Drive played a pivotal role in organizing and storing relevant working documents, including meeting minutes and assignment reminders. This centralized repository ensured accessibility to essential project information and enhanced overall team coordination. By leveraging structured division of work and utilizing diverse collaboration platforms, our team maintained momentum, promoted consistent communication, and facilitated seamless collaboration throughout the project lifecycle.

Challenges

Our project encountered several significant challenges during the development of SitesMobile. First and foremost was relearning GitHub and understanding its mechanics, such as pushing, pulling, and creating branches, proved to be a steep learning curve. Integrating GitHub with Xcode and navigating the Xcode interface added another layer of complexity to our workflow. Additionally, resolving non-automatic merge conflicts, particularly when pulling changes from the main branch, required careful attention to maintain project integrity. Task management became critical to avoid overlapping responsibilities, while alterations to project file locations and names required careful attention to prevent disruptions. The use of the Firebase API, especially in implementing login functionality and understanding its data handling processes, presented further challenges. Conceptualizing and understanding the intricacies of NoSQL data relationships and decoding data from Xcode accurately were essential yet intricate tasks. Implementing asynchronous functionality and utilizing closures were technical skills we first had to learn to understand and then utilize in our development process. Lastly, organizing project files effectively became crucial for streamlining collaboration and ensuring project cohesion.

Adjustments

In response to the challenges encountered, our team implemented several key adjustments. To enhance our team’s proficiency with GitHub and ensure effective collaboration, the team watched tutorial videos that were specifically focused on working with GitHub through Xcode. The group also studied best practices for version control, which included branch naming, management, and conflict resolution strategies. Thus equipping each team member with the skills necessary to contribute more effectively to our project's codebase.

Recognizing the importance of task management and effective file organization, we set specific guidelines for file naming and updates to project locations. To enhance coordination and clarity of responsibilities, we utilized GitHub Projects to manage tasks, ensuring a well-structured workflow that effectively prevented overlaps in team assignments. Faced with the complexities of Firebase and asynchronous operations, the team watched tutorial videos to deepen our understanding of NoSQL databases and master asynchronous patterns and closures, leading to a more responsive and scalable application. Additionally we extensively refactored our codebase, incorporating enums, generics, and property wrappers to enhance code clarity and maintainability.

To address the challenge of external API integrations, such as Firestore, we made sure to check for and pull any new updates for the packages used in the app. These adjustments were pivotal in navigating the obstacles we encountered and ensuring the continued progress and effectiveness of our project within the planned timelines. Additionally, at the tail end of development and testing, we encountered challenges where we reached the limit on how many reads we could use per day towards the database, highlighting the need for more efficient resource management. At the time we were able to upgrade to the paid plan in order to continue working on the app. However, it can be noted that the issue of budget may not prove to be as significant a problem at larger scales of implementation, such as within a university setting, and can be mitigated through optimization efforts.

Ethics

Throughout the development of SitesMobile, our team encountered several ethical considerations that were integral to our project's success. These ethical dilemmas revolved

primarily around the responsible tracking and management of users' data, particularly in relation to location services and employee monitoring.

One significant ethical consideration pertained to the responsible use of location tracking. The application offers real-time location tracking services to facilitate seamless navigation for employees between different sites. However, ethical concerns arose regarding the storage and utilization of specific location data. To address this, we considered storing location data only for essential purposes, such as verifying employee presence during clock-in processes. However, we ultimately decided against this option. Additionally, even without storing location data, we had to consider the active use of location services by our application in order to utilize the maps functionality. We made sure to implement Apple’s feature that makes it so that users will have control over their privacy settings, with the ability to enable or disable location services at any time.

Another ethical consideration focused on mitigating the potential misuse of application functionality. Specifically, we identified the possibility of employees falsifying cleaning reports to manipulate work records. While this issue predates our application, our team had to take this into consideration throughout development. As stated above, though, we ended up deciding against tracking employee location for clock-in and clock-out verification, which in turn led to us rejecting the proposed option of ensuring employees are located at a certain site to submit any cleanings. While there could have been some benefits to combatting potential employee dishonesty, we feel strong in our decision to prioritize users’ privacy especially given the potential issue was preexisting.

Additionally, ensuring fairness in data monitoring and interpretation emerged as a critical ethical concern. Beyond location data, SitesMobile stores information related to employee performance and compliance with work standards. This caused ethical considerations to arise concerning the potential biases of managers and employees in interpreting this data, as well as the potential for employee discipline based on said data. Some useful advice in this situation comes from Middle Georgia State University professor Johnathan Yerby in his research entitled "Legal and ethical issues of employee monitoring" in 2013. Yerby suggests that employers do the following to ensure fair tracking of data: "[notify] the employee, give the employee access to information being collected, and give the employee the right to dispute evidence". In our case, our app will let employees know they are being tracked location wise, as stated above, and they will be aware that they are submitting cleaning reports. If they wish to see the information, they can reach out to managers who can access data tracked on their manager facing view of the app. And lastly, it is important that the employees are able to dispute in any of those given situations of disciplinary action.

Results

Throughout this development process, our team maintained focus on the end goal: creating an application that was functional and to be proud of. Despite numerous challenges, lessons, and considerations along the way as described above, there was always a group mindset that narrowed in on delivering a product. As the project is now completed, we have discussed the results of the final iteration as well as the testing involved in producing them below.

Testing Process

Throughout the development of SitesMobile, the team conducted testing to ensure the functionality of the application. A significant portion of the testing went through the simulator and preview functions built into Xcode, allowing the development team to test the functionality and usability of the code they had created as soon as they believed the view was functional. In addition to individual testing, the team also conducted group testing sessions following each major milestone in the development process, like specific form submissions were completed as well as the map being implemented. The group testing sessions allowed the team members to collaborate, identify any issues or consistencies, and make sure the app functioned across different devices. Katie also frequently built and tested the app onto her personal iPhone to validate its performance and usability on a physical iPhone.

In week 13 of the project, the team achieved a significant milestone as the app reached a state where it was nearly fully functional for employees.The core features such as user authentication, site navigation, inventory management, and issue reporting, were implemented and tested. However, there were a few remaining tasks to ensure the app's readiness for real-world use. These tasks included filling the database with accurate and up-to-date information about the various Computing Sites and computers available at each location. Once these final data-related tasks were accomplished, the team felt confident in deploying the app to a select group of Computing Sites employees for real-world testing.

The feedback we received from the testers was primarily positive. They appreciated the app's appealing design and found the map function very useful, as it displayed the current location and nearby sites. The ability to filter the map by different site groups was very well-received. They noted that some specific sites either lacked data, or some were missing as a whole, but we were still in the process of inputting data at the time.

Some constructive feedback we received was how the inventory submission form was slightly confusing without any prior explanation. In the existing Microsoft Form, users typically enter a number in the text box to indicate the count of the specific inventory. In the app, the team used, “count,” “confirm,” and “fix” options. The count is represented by the believed quantity at that site, and users could toggle the “confirm” option to verify the count or use the “fix” option if there was a discrepancy in the inventory. The new approach required some clarification for user’s used to the previous system. Another aspect of confusion was our initial submit boxes. Initially, we provided the “Confirm and Exit” for cases where no supplies were used and “Confirm and Continue” when someone did use a supply”. Based on user feedback saying that it was a bit confusing on communicating which button did what, we changed the options to “No Supplies Used” and “Supplies Used” for better clarity.

Overall, the testing process of the app provided valuable insights into usability and the functionality of SitesMobile. It allowed us to gather feedback from real employees, where they identified areas for improvement, and made necessary adjustments to enhance user experience.

User Experience

In the initial stages of developing views within the Xcode environment, our team navigated the delicate balance between achieving basic functionality and ensuring a seamless UI experience. While our initial views maintained simplicity, we prioritized their effectiveness to establish a sturdy foundation for subsequent iterations. Given the intricate nature of our application, which encompasses upwards of 21 interconnected views, our first iteration leaned towards a more conventional design to ensure coherence and functionality.

As we progressed to our current iteration, our UI inspiration drew from two primary sources: the style of the University of Missouri’s Division of Information Technology (DoIT)

program and a simplistic, tech-oriented design. While we aimed to produce creativity into elements such as the logo and front page, we placed emphasis on the use of recognizable, clear icons, and labels to facilitate easy navigation for employees at Computing Sites. Considering the core objective of our application to streamline work processes for Site employees, we maintained a balance between simple yet visually appealing interfaces and usability, adhering to UI/UX (user experience) guidelines from previous courses and reputable resources such as the Web Content Accessibility Guidelines (WCAG). Xcode's capability to implement UI effectively and straightforwardly gave our UI development a slight advantage.

This overall approach is evident in several design choices, including a clear hierarchy of information, descriptive icons, and the utilization of a TabView layout, which is a familiar interface pattern for smartphone users. For example, our adoption of the TabView layout offers intuitive navigation, ensuring ease of access to various functionalities within the application. Furthermore, the incorporation of recognizable visuals, such as clear icons and labels, enhances user comprehension and usability.

As stated prior, we engaged a sample of users comprising student employees and managers from Mizzou’s Computing Sites to gather feedback on usability, functionality, and overall user experience. Feedback from users highlighted areas of strength, such as the intuitive navigation, clear hierarchy of information, and the effectiveness of features like live location tracking and inventory management.

In one instance, our commitment to accessibility encountered a challenge when some of our contrast ratios did not meet the recommended standards (see Figure B9). As outlined in the Web Content Accessibility Guidelines (WCAG), "The visual presentation of text and images of text has a contrast ratio of at least 4.5:1." This shortfall prompted us to reassess our design

choices and make necessary adjustments to ensure compliance with this crucial guideline. By addressing these issues, we aimed to enhance the readability and usability of our application for all users, considering varying levels of visual ability.

Furthermore, while striving to uphold accessibility standards, we recognized the comprehensive accessibility features integrated into Apple devices. These built-in features cover a wide range of accessibility concerns. Leveraging these features not only aligns with our commitment to inclusive design but also ensures that our application is accessible to a broader audience, including users with diverse needs and preferences. By embracing these native accessibility features, we reinforce our dedication to creating a user-friendly experience that is accessible to all individuals, regardless of their abilities or limitations.

Elements of Security

Throughout the development of SitesMobile, our team encountered several security considerations that were crucial to ensuring the protection of sensitive employee and workplace data. The primary security concerns revolved around the secure storage of information such as employee details, access to different Computing Sites, and location data. We recognized that if this data were to be compromised or misused by a manager, it could lead to serious consequences like privacy breaches, unauthorized access, or even identity theft.

One critical area of focus was the secure storage of user credentials and controlling access to the application. According to a Data Breach report by IBM (2023), "The Global average cost of a data breach in 2023 was 4.45 million US Dollars." To mitigate the risk of a costly data breach, we made the decision to store employee information such as email, name, and work information such as their student ID number, including university email, pawprint, and other work-related details, in Firebase. Firebase is a secure cloud-hosted database platform

developed by Google that offers robust features for protecting user data. As stated in Firebase's documentation (Firebase, 2024), data security is achieved through "Firestore Security Rules to handle serverless authentication, authorization, and data validation. For apps that use Cloud Storage, to define conditions for access to your Cloud Storage resources in database documents." By leveraging Firestore's in-server ruleset, we were able to securely limit access to the app, ensuring that only authorized individuals could retrieve and manipulate data.

When an employee signs into SitesMobile, they do so using Google's Sign-In service. After entering their university email, they are redirected to the University of Missouri's Single Sign-on page, where they must complete the user multi-factor-authentication process. By integrating Google's Sign-in, SitesMobile takes advantage of the university's single sign-on functionality for authentication. This approach ensures that the user's university password remains protected by the University's authentication system. Once a user successfully signs in, their pawprint and university email are securely stored in Firebase. Most importantly, their University password is not saved in Firestore, as the password has already been authenticated through the Single Sign-on process.

Another significant security risk we carefully considered was the potential for a data breach involving the University of Missouri's systems. Since SitesMobile relies on university login credentials, it is inherently susceptible to any data breach that the University of Missouri may experience. In August 2023, the University of Missouri's system suffered a data breach in which an unauthorized third party gained access to a university employee's email account and copies of student records. Although SitesMobile does not directly store or access university system accounts, a compromise of the university's systems could potentially expose credentials and allow unauthorized access to SitesMobile and its stored data.

Conclusion

The experience creating SitesMobile bought our team invaluable experience in start-to-finish application development. More specifically, our team was able to gain experience in the areas of project and team management, overall teamwork, as well as in the more technical aspects of coding and manufacturing a mobile application. The finished product is complex, functional, and put together in the most efficient way our team could produce given the time constraints.

Having said that, there was a learning curve on the road to completion. Our team had never dealt with a project of this scale before. Due to this, it is important for us to note that there are areas for improvement. These include but are not limited to overall cleanup and refactoring of our code and database, reducing redundancy, adding functionality for admins to be able to edit site data in-app, and making our UI less Apple-design-forward and more in line with other University of Missouri applications. Despite these potential areas of improvement, our team utilized our time wisely and in the most efficient manner we could manage. With twice weekly meetings, multiple group chats, and an organized filing system, we were able to stay on track as well as meet and exceed our minimum viable product. Conclusively, our developmental team worked hard to churn out a project that could improve the University of Missouri Computing Sites' overall experience, and we feel this expectation was met and exceeded.

References

CollegeNET 25Live. (n.d.). *Event & Space Scheduling.* https://25live.collegenet.com/pro/missouri#!/

Firebase. (n.d.). Get started with Cloud Firestore Security Rules. Firebase Documentation. Retrieved April 7, 2024, from https://firebase.google.com/docs/firestore/security/overview

IBM. (2023). Cost of a Data Breach 2023. https://www.ibm.com/reports/data-breach

Pricing - When I Work. (n.d.). When I Work. Retrieved April 7, 2024, from https://wheniwork.com/pricing#:~:text=Over%20one%20million%20people%20in

Restricting Where Users Can Clock In Or Clock Out – When I Work Help Center. (n.d.). When I Work Help Center. Retrieved April 7, 2024, from https://help.wheniwork.com/articles/restricting-where-users-can-clock-in-computer/

Yerby, J. (2013, January). Legal and ethical issues of employee monitoring. ResearchGate. https://www.researchgate.net/publication/313656700_Legal_and_ethical_issues_of_employee_monitoring

Understanding Success Criterion 1.4.3: Contrast (Minimum) | WAI | W3C. (n.d.). https://www.w3.org/WAI/WCAG21/Understanding/contrast-minimum.html

University of Missouri. (n.d.). *Brand & Identity: Colors.* https://identity.missouri.edu/visual-identity/colors/

University of Missouri Division of Information Technology. (n.d.). *Classroom and Computing Sites.* https://doit.missouri.edu/services/computing-sites/

Appendix A

Stages of Early Development

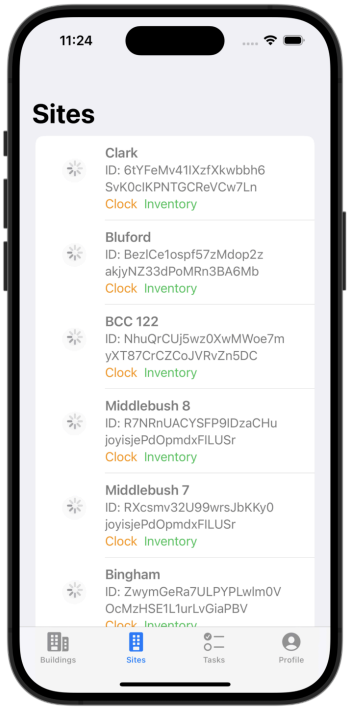


Figure A1. First iteration of loading a dynamic list of Computing Sites where the secondary information has yet to be loaded in full detail.

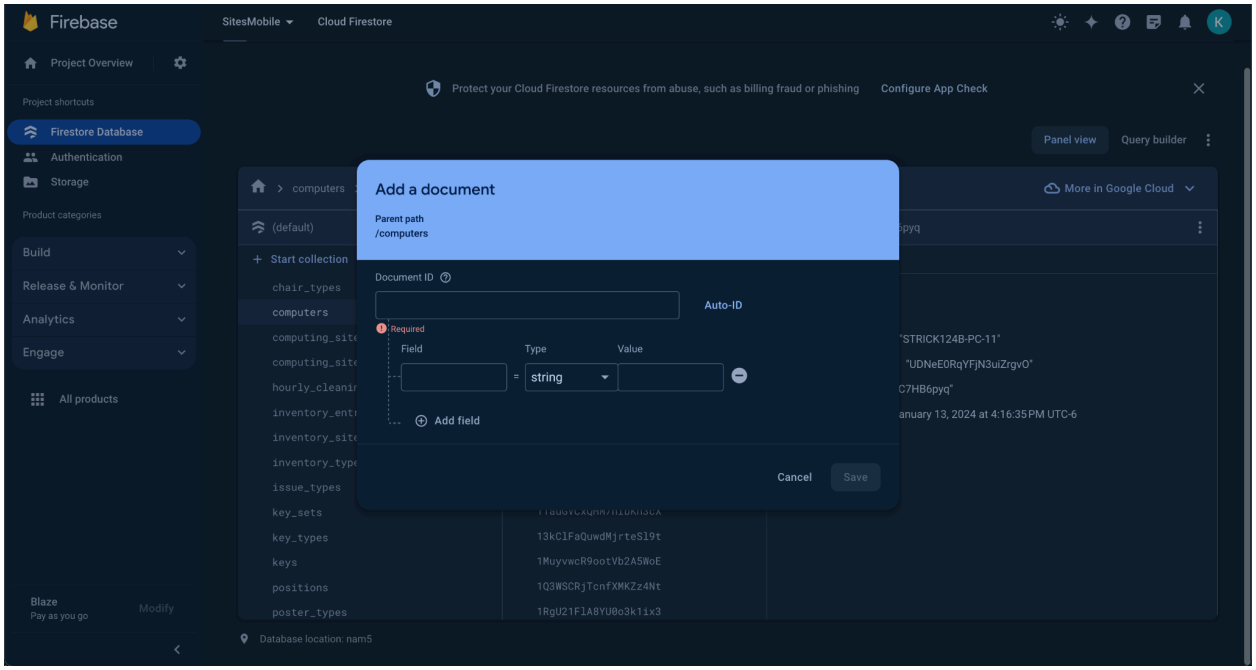
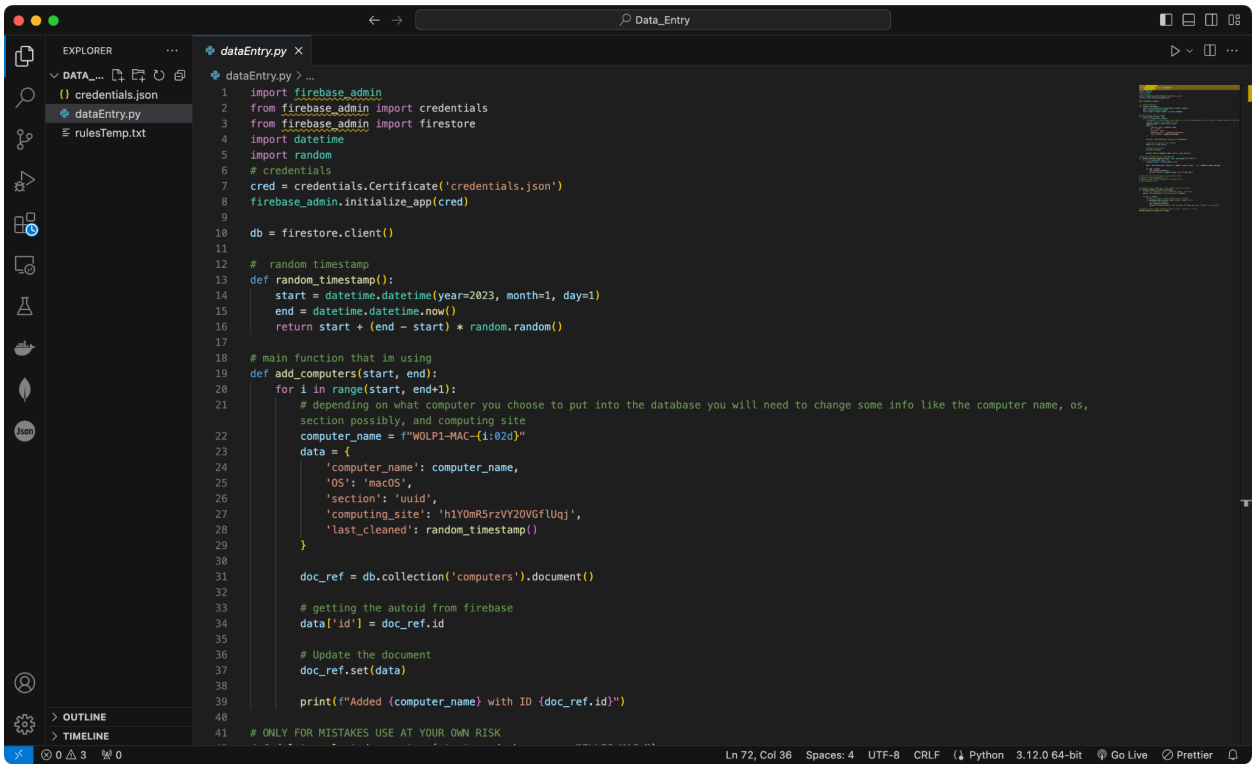


Figure A2. Manually creating a computer document in the Computers collection using the Firebase Console.




```python
import firebase_admin
from firebase_admin import credentials
from firebase_admin import firestore
import datetime
import random
# credentials
cred = credentials.Certificate('credentials.json')
firebase_admin.initialize_app(cred)

db = firestore.client()

#  random timestamp
def random_timestamp():
    start = datetime.datetime(year=2023, month=1, day=1)
    end = datetime.datetime.now()
    return start + (end - start) * random.random()

# main function that im using
def add_computers(start, end):
    for i in range(start, end+1):
        # depending on what computer you choose to put into the database you will need to change some info like the computer name, os,
        section possibly, and computing site
        computer_name = f"WOLP1-MAC-{i:02d}"
        data = {
            'computer_name': computer_name,
            'OS': 'macOS',
            'section': 'uuid',
            'computing_site': 'h1YOmR5rzVY2OVGflUqj',
            'last_cleaned': random_timestamp()
        }

        doc_ref = db.collection('computers').document()

        # getting the autoid from firebase
        data['id'] = doc_ref.id

        # Update the document
        doc_ref.set(data)

        print(f"Added {computer_name} with ID {doc_ref.id}")

# ONLY FOR MISTAKES USE AT YOUR OWN RISK
```

Figure A3. Python program dataEntry.py that supplemented data entry into Firestore.

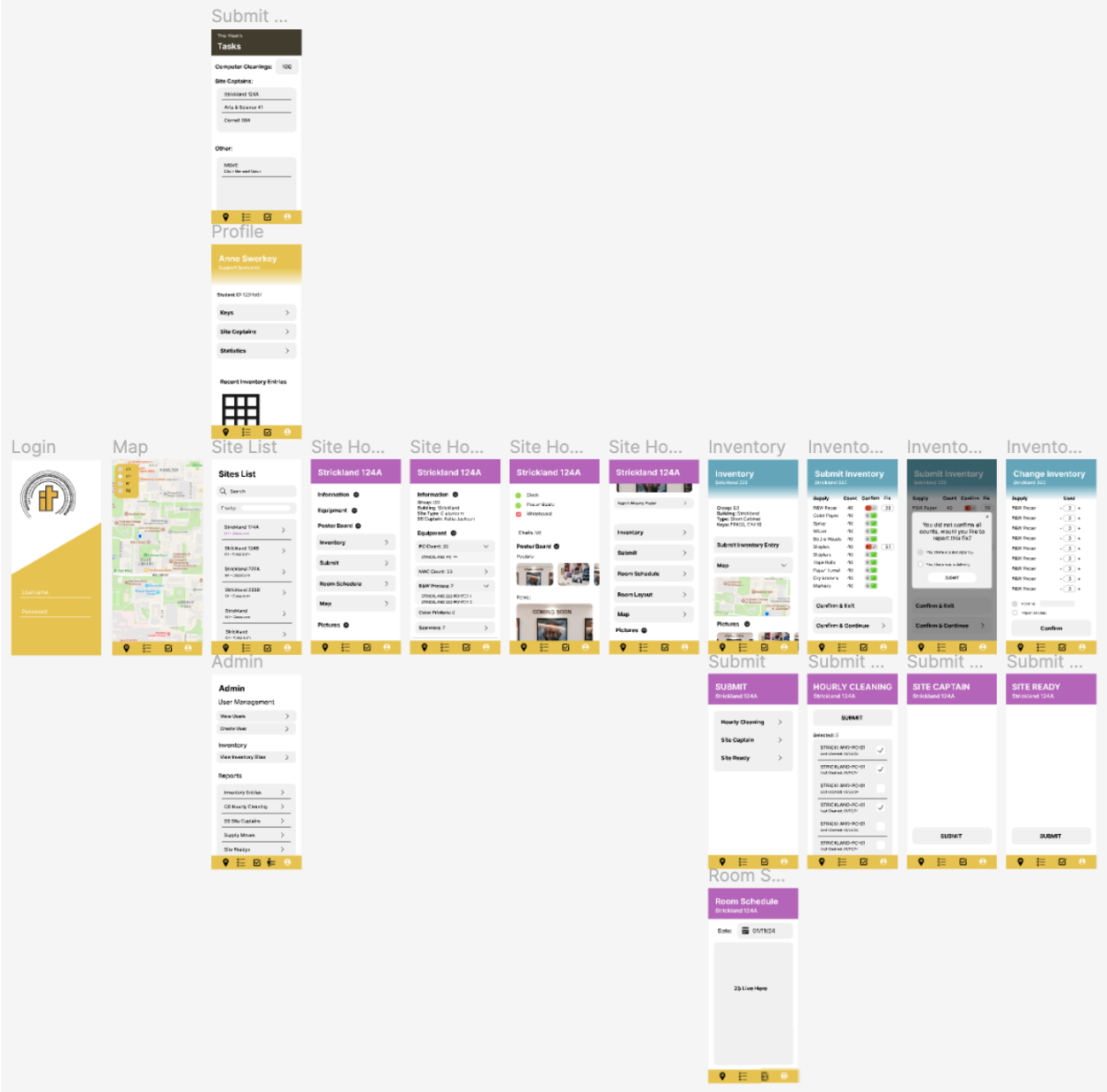


Figure A4. The wireframe of SitesMobile that guided development through Phase 2: Design and Discovery.

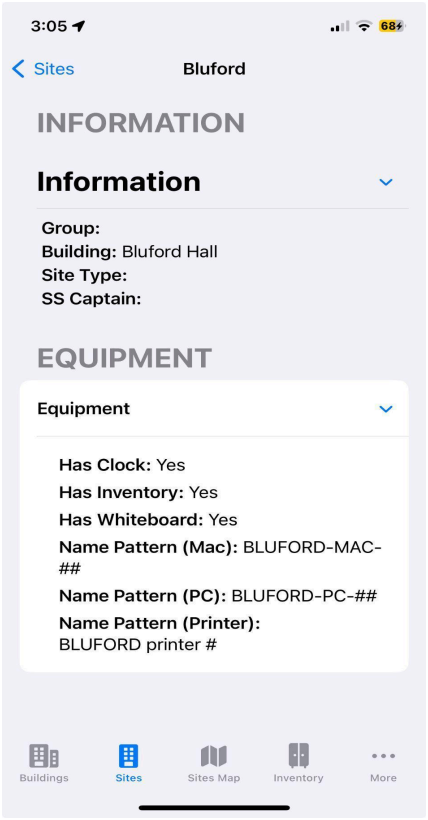


Figure A5. The rudimentary version of the DetailedSiteView.

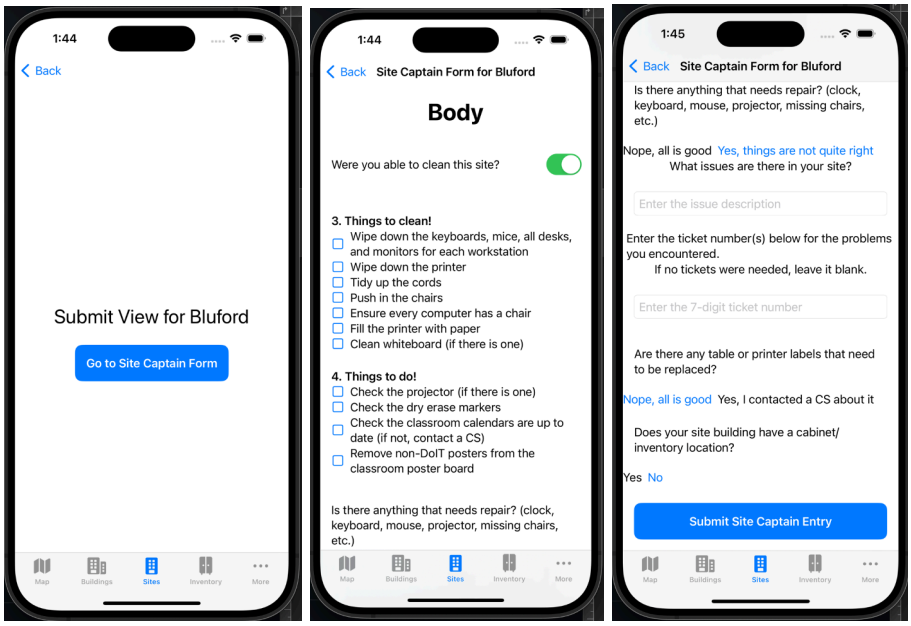


Figure A6. The first version of the Site Captain form.

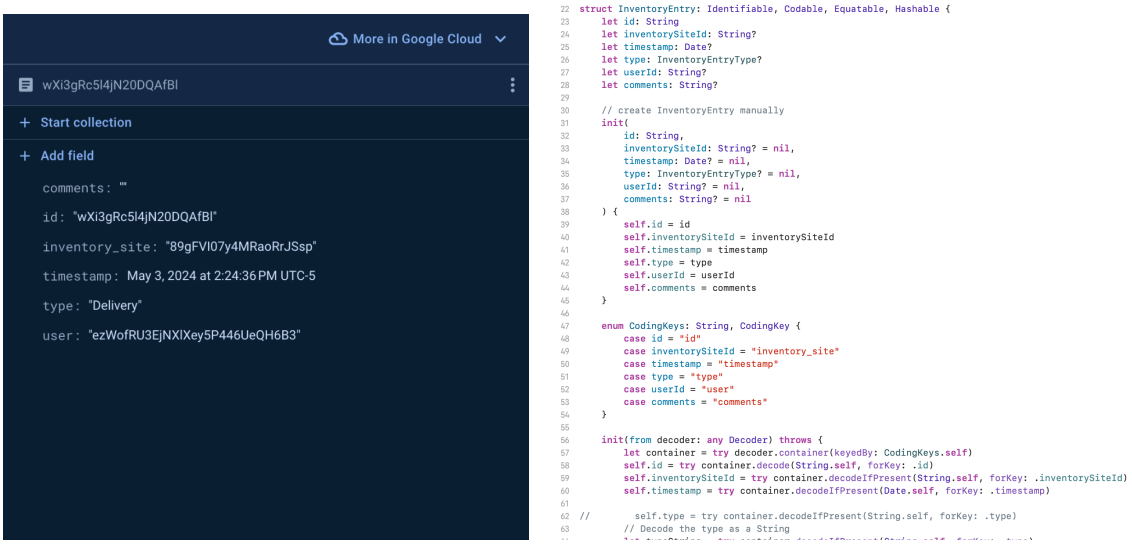


Figure A7. Image of the document structure for an InventoryEntry in Firestore on the left and how it is decoded in the SiteMobile app using a manager file on the right. Note that the coding keys and data types must be consistent between the two, or the data will not be pulled.

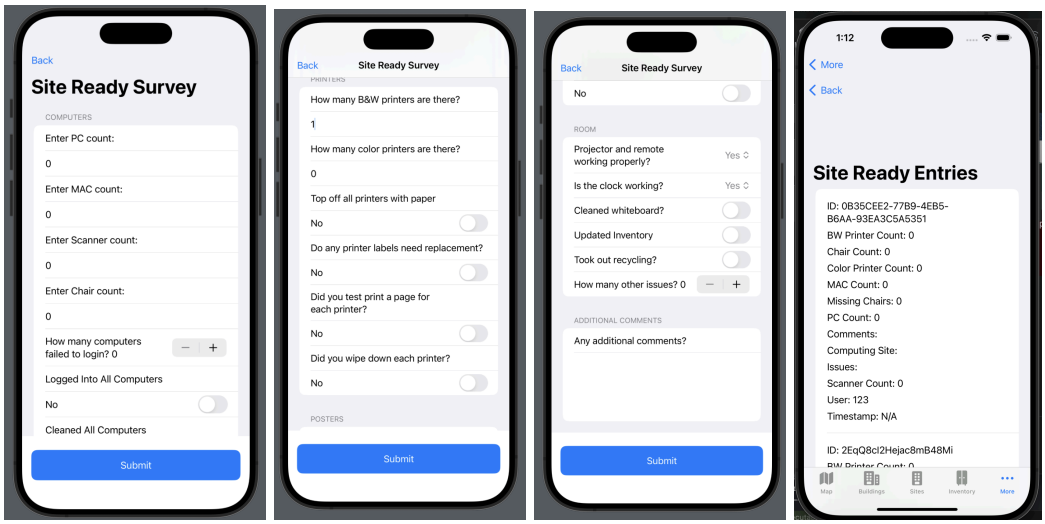


Figure A8. The early version of the Site Ready form (left three images) and SiteReadyEntriesView (rightmost image) before refactoring.

Appendix B

Final Iterations

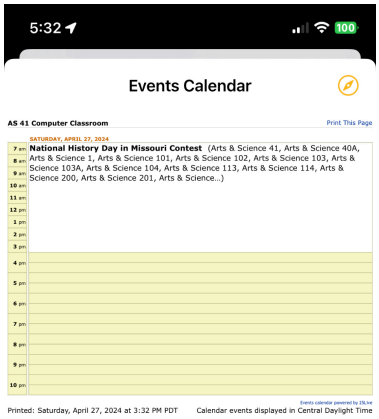


Figure B1. Calendar popup within he DetailedSiteView for classroom sites. The icon on the top right will redirect the user to their default browser app.

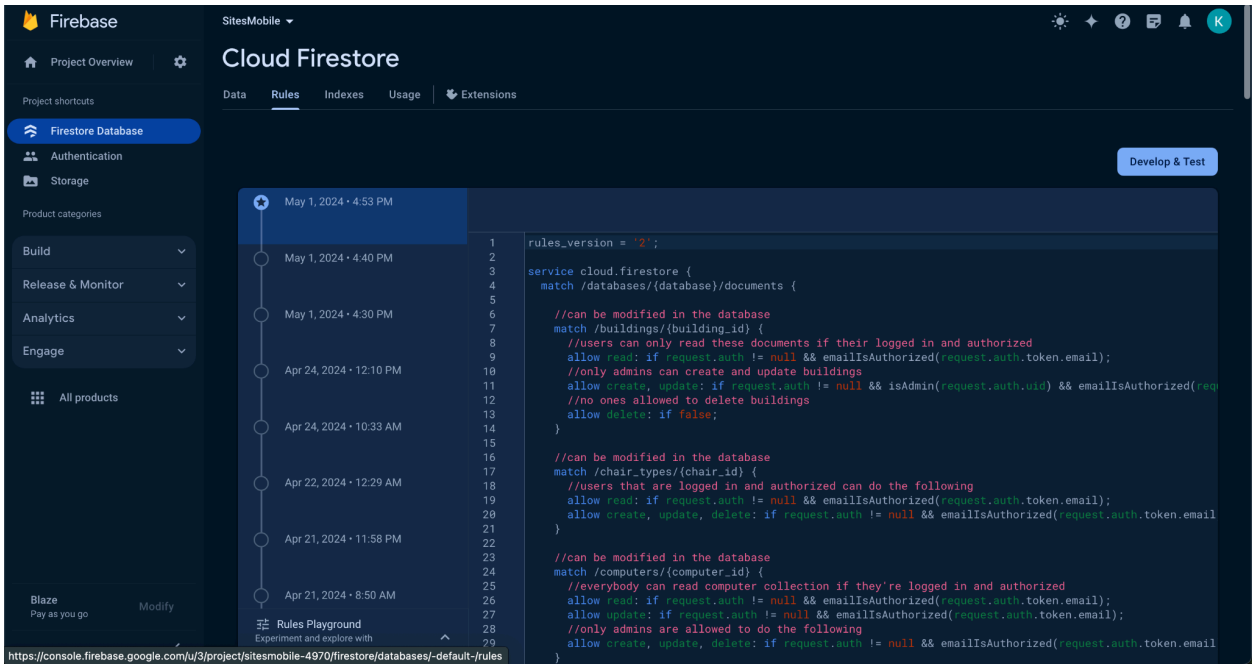


Figure B2. Firestore security rules in FIrebase Console. These rules stipulate conditions that requesting users must pass to have certain read, write, and delete access for each data collection.

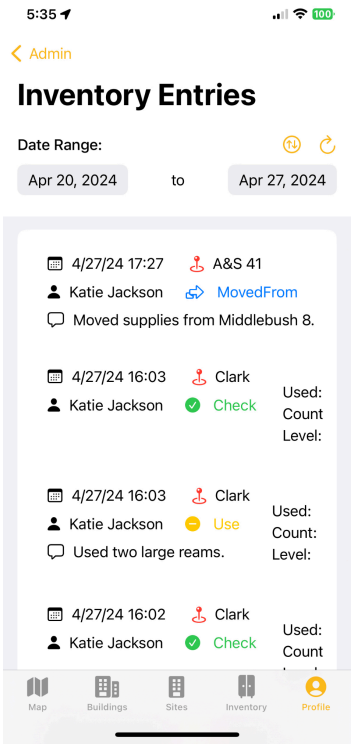


Figure B3. InventoryEntriesView

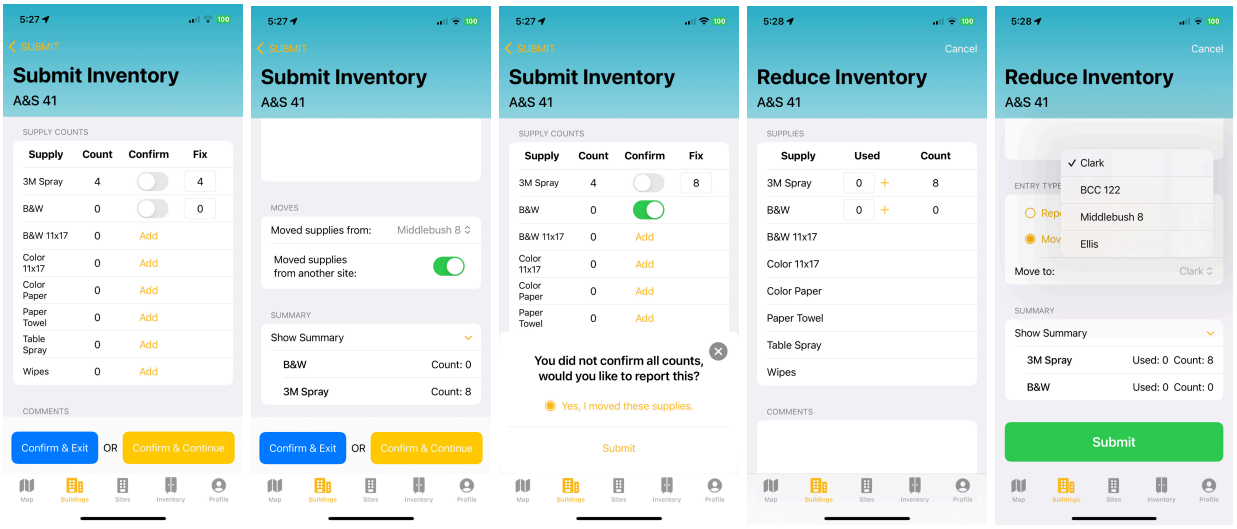


Figure B4. Inventory Submission sequence

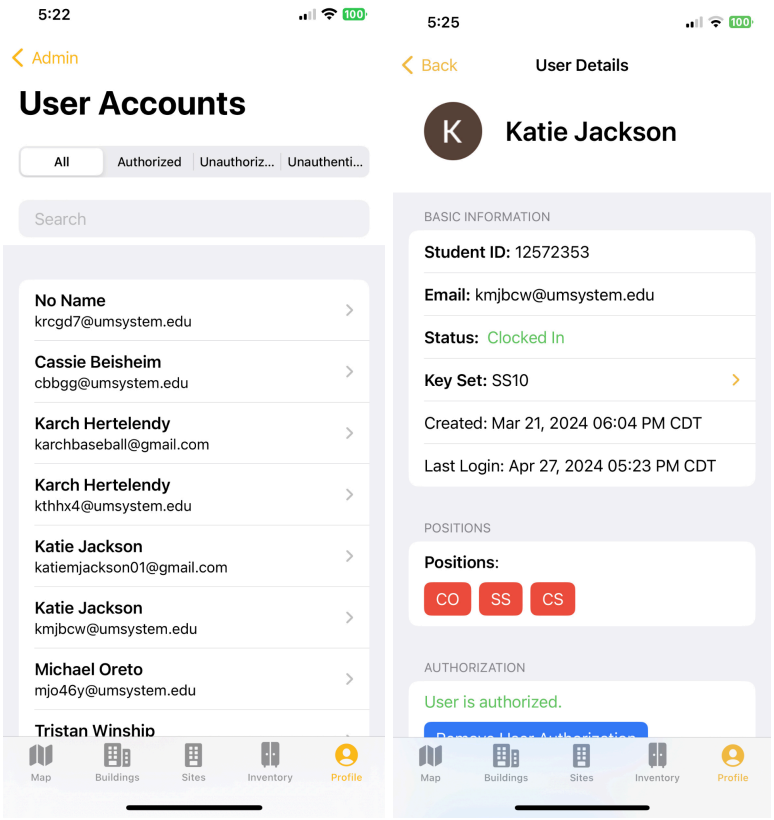


Figure B5. Admin and user profile view

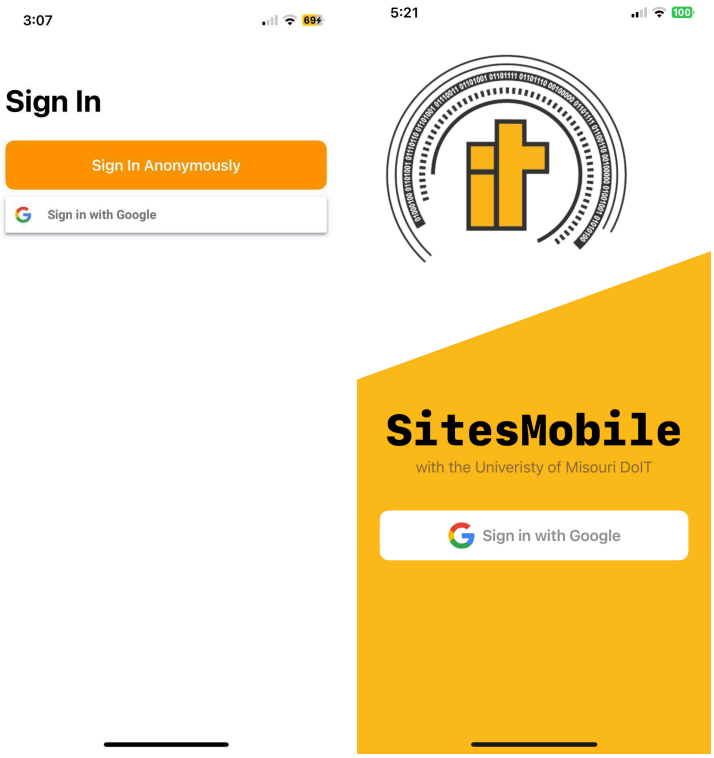


Figure B6. The original login page (left) and the updated log in page (right).

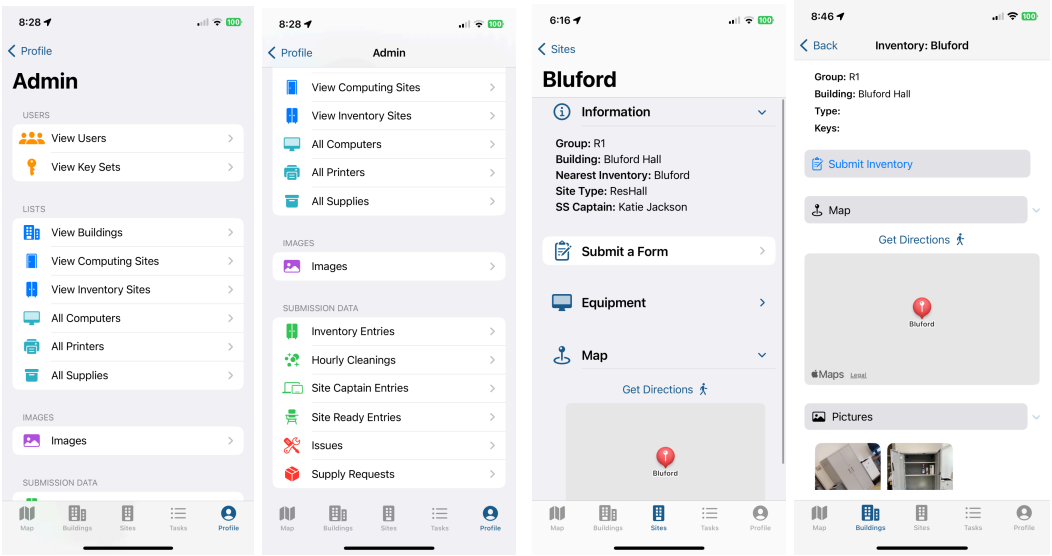


Figure B7. The finished AdminView, DetailedSiteView, and DetailedInventorySiteView respectively.

Figure B8. The original MapView (on the left) and the updated group filter for the MapView (on the right).

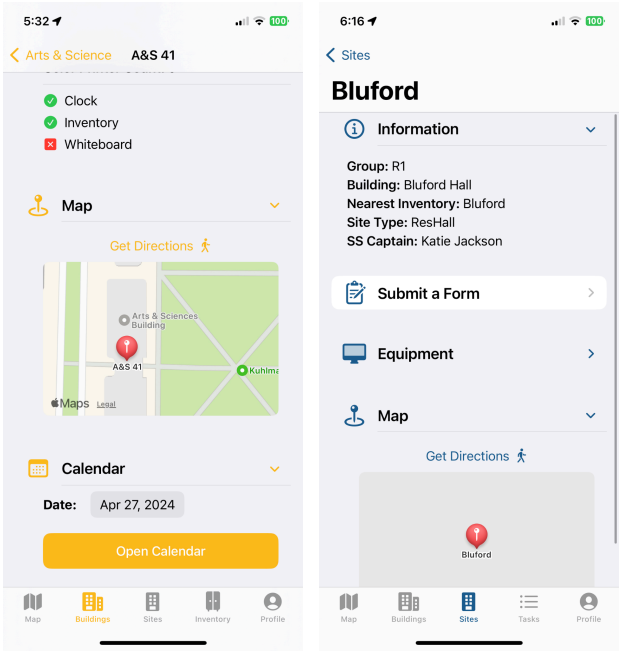


Figure B9. The accent color in light mode had been changed from MU Gold (left image) to Blue (right image) to comply with the recommended contrast ratio for accessibility. Both colors were derived from University of Missouri's brand colors (University of Missouri, n.d.).