University at Albany
Department of Computer Science

ICSI 490 Internship Report

May 2021

Christopher Pouch
(UAID 001407678)
B.S. in Computer Science & Applied Mathematics (D.A.)

Telescope Casual Furniture

Introduction

Telescope Casual Furniture is an international furniture company based in Granville, NY. Telescope designs and builds a wide variety of furniture pieces from raw materials to finished product, with a large multifaceted factory to accommodate every step of production.

This report details my software development internship with Telescope's IT department. The IT department has developed in-house systems to improve factory operations and help solve problems and inefficiencies. With multiple departments (sewing, welding, metal, etc.) in continual operation, new problems and tasks come up on a rolling basis and their solutions can have significant effect on the everyday operation of the company.

The purpose of this report is to explain the internship experience from start to finish. It will cover the goals and requirements of the internship, an overview of my work, a detailed technical explanation of my main project (a radio management system), and analysis of the experience and learning outcomes.

Job Description

The basic requirements of this internship were to do programming work that would benefit the company, and to learn and improve my own skills through application, all under the guidance of more experienced team members. To do so, the expectation was for me to take responsibility for all aspects of my work, including business logic, technical design, implementation, and testing. I was expected to be a reliable member of the team, attending meetings, communicating effectively, and developing an understanding of the overarching plans in which I played a part.

It was expected that I would work primarily as a software developer, contributing to software projects as business needs and timing dictated. As part of the development process, I was expected to get familiar with Telescope's technology stack, ask questions, and make sure I understood all project requirements.

The IT department promoted my chances of success by arranging for me to design and implement solutions with a high degree of autonomy, responsible not only for the good ideas, but also for dealing with the consequences of bad ideas. It was understood that if I ran into trouble there would be assistance available, and that there would be opportunities for co-working and teaching.

Specific Tasks

My main task, which was ongoing throughout the whole internship period, was to create a management system for portable radios. The radios were previously targets of theft and people kept losing track of them, so the goal was to allow employees to check radios in and out by scanning their labels with a barcode reader. With user information available to supervisors, it would be possible track who had missing radios last, when they were last scanned, etc.

This project was to be built into the existing Telescope codebase, which offered some existing functionality relevant to the radio system and connected to other parts of the technology stack. The logic and design of the radio system were to be made from scratch. The goal was to get the radio program to be fully functioning and roll it out for factory use, so in addition to designing and coding, it was necessary to understand the nuances of use cases and perform good testing.

In addition to the main project, I was part of weekly IT department meetings. Telescope is in an ongoing transition to Odoo ERP, a scalable resource planning platform, and weekly meetings were arranged with an Odoo representative to go over technical questions and instruction. Periodically there were general department meetings to discuss current projects and priorities.

Detailed Project Description

The radio management system was my primary project at Telescope. I worked on it from start to finish, beginning with a specifications briefing and eventually rolling out the completed project for factory use. This section of the report discusses the project in detail in several sections.

STATEMENT OF PROBLEM

Due to a lack of accountability and tracking of users, there were issues with portable employee radios being lost and stolen. The purpose of my project was to develop a system that employees would use to check radios in and out, one that would keep track of who had which radio, how long they'd had it, when it was last scanned, etc.

The project responsibilities were divided between myself and my business mentor, who was in charge of the database side––creating PHP-wrapped SQL functions and adding relevant fields to existing tables. The business mentor was also available for questions and advice whenever needed. My responsibility was the C# code, including the functionality of the radio system and the necessary checks, testing, and documentation.

DEVELOPMENT ENVIRONMENT

The radio project was built within an existing resource planning system that was developed in-house with C# in Visual Studio. Visual Studio's Xamarin platform

provided the skeleton of the user interface code, supporting drag and drop for common widgets like buttons and grids. The backend was written in C#. Telescope's Postgres database was accessed via an intermediate PHP website where wrapped SQL functions were stored for use by the C# system. Touchscreens running on Raspberry Pi 3 computers were placed throughout the factory and ran the software for daily use by employees.

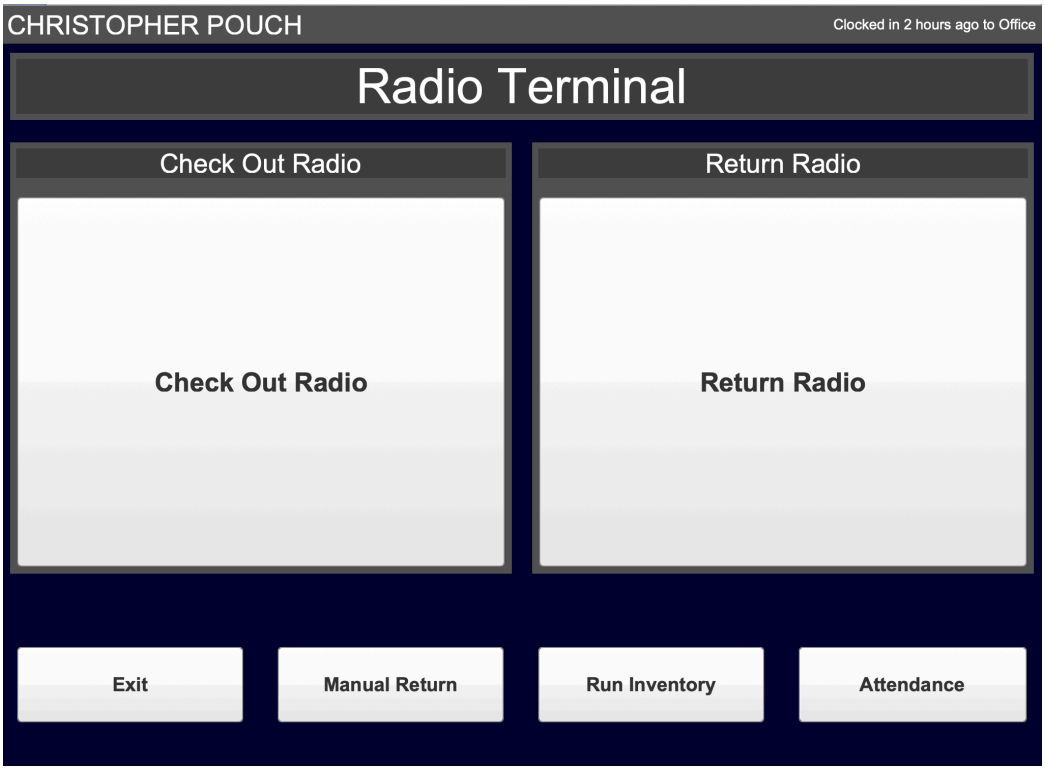
RESEARCH APPROACH

My first research requirement was to learn and understand the existing C# system which my radio program was based on. The pre-existing codebase powered the company's payroll calculations, factory inventory flow, etc. Understanding that system took a lot of time and progress was initially very slow, but eventually I hit a tipping point and my understanding accelerated. I paid attention to functions that would be relevant to the radio project in the future, and took note of the design patterns that I would later emulate during development. In addition to the C# code, I had to get familiar with some other parts of Telescope's technology stack. Raspberry Pi 3s, for example, were new to me, as was Xamarin Studio for Visual Studio. All the above helped me get ready for the programming and also helped me understand the overarching design of the system.

SOLUTION

Design overview

The design of the radio system centered around two main screens, RadioHomeScreen and RadioInventoryScreen. Users could view and interact with one screen (buttons, functionality, etc.) and switch to other screens. Because the pre-existing codebase had a lot of useful infrastructure already--classes to keep track of employee information, screen interface with header template and timeout functions, etc.--most of my code went directly in the files associated with the two main radio screens. Within these two screens, there were three main functionalities: users could check out radios, return radios, and, if they were supervisors, view an inventory report on all radios.



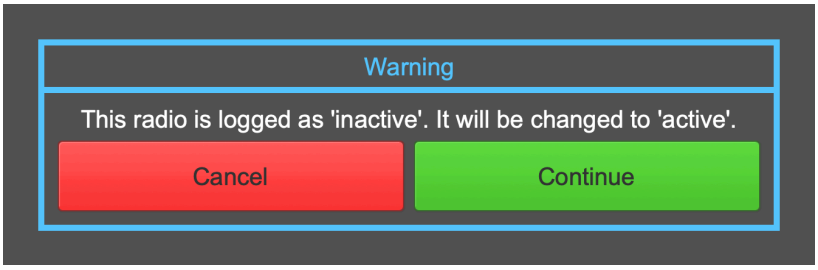
RadioHomeScreen, one of the two main screens for the project.

Technical design

The radio system was added to the existing codebase as a 'mode option'. The system could run in normal mode, with no radio functionality, or it could be run in radio mode where a "-R" parameter would tell the program to load the radio screens. Both of the radio screens relied heavily on database queries, which supplied a list of radios, employees, and information about them. The Telescope Casual database was accessed via a PHP website that acted as an intermediary. SQL functions on the PHP site were called by the C# code, then they queried the database and returned the results to the C# code, acting as an object-relational mapper. The program behaved according to this information, which was reloaded every time the application started or a database operation was successfully done. For example, when a user scanned their badge to sign in to the system, their personal information would be loaded into an InformationSummary class. That class would contain a boolean variable 'assigned_radio'. When RadioHomeScreen loaded, it would check 'assigned_radio', and determine whether to display a "check out radio" button or hide the button and display a message (see right).

It was a business decision to allow only one radio per person.

For each attempted user action, many checks had to go on in the background. If a user tried to return a radio, it mattered whether or not they were the one to check it out in the first place and what state the radio was in (active, missing, stolen, inactive). In unusual cases, like when someone tried to return another person's radio, confirmation would be required (see below) and the action would be logged to the database. With several relevant variables, the number of potential user situations was large and it was important to create defined rules to apply to many cases rather than define behavior for each case individually.

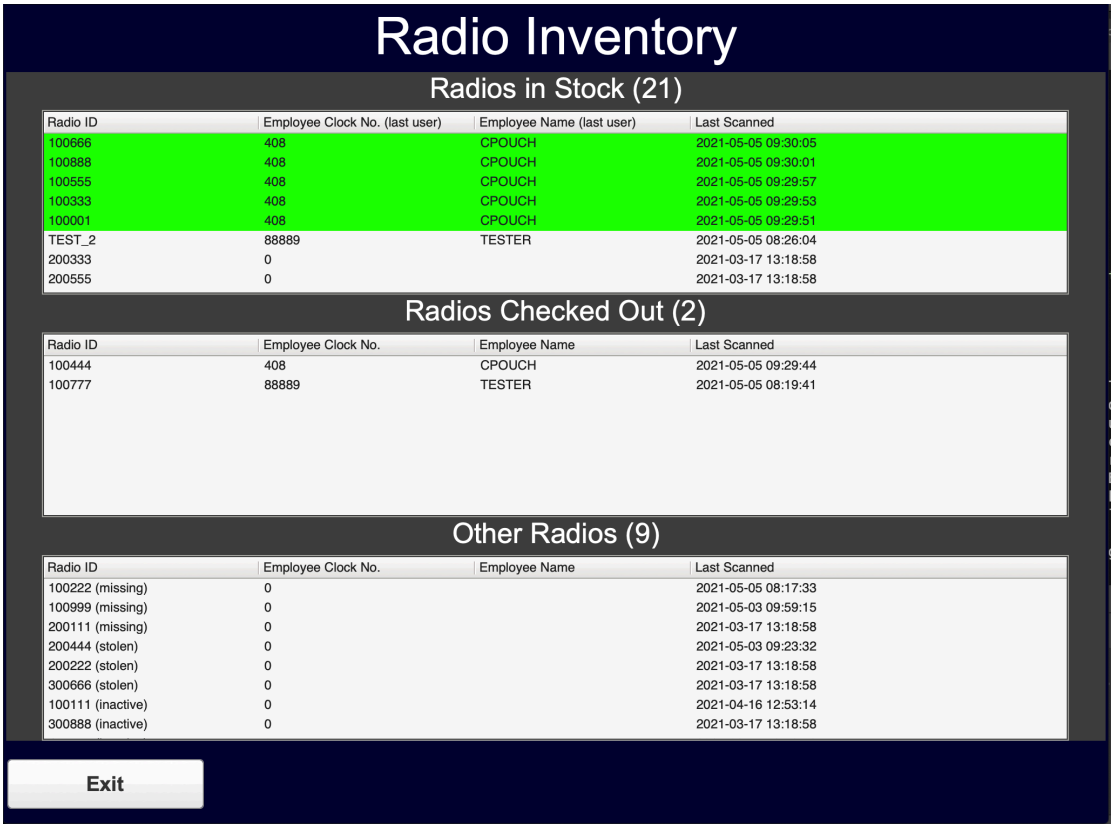


Use cases deemed unusual required user confirmation.

The user interface for the system was based on a combination of Xamarin Studio's drag and drop widget options with the GDK-Sharp widget library. The skeleton of each screen was built with Xamarin and the dynamically created elements were made with Gdk#. The advantage of Xamarin was that all those manually-added widgets existed at the class level and could be accessed by all class functions. Xamarin auto-generated files to represent this formatting and they were stored out of sight and out of mind. When those widgets were created inline with Gdk#, they were only accessible at the class level if a manual declaration was added. This could get messy,

given how many containers and subcontainers went into creating widgets. The advantage of dynamically generating widgets with Gdk# was flexibility: the program could generate X number of buttons based on conditions, whereas with the Xamarin drag-and-drop that number had to be chosen upfront.

The RadioInventoryScreen was designed to show a complete list of all Telescope Casual radios, and allow supervisors to easily keep track of them and notice any problems. The inventory screen had three tables to represent in-stock, checked out, and 'other' radios. At the end of the day, all radios should have been back at the station, so the inventory screen allowed users to scan all radios and highlight their presence or lack thereof. When a radio was scanned (or 'verified'), its row would turn green. So if all the in-stock rows turned green, all the radios were there.



As my code grew in size, it became clear that the level of organization would have to follow suit. It was necessary at several points to refactor and make the code more modular. One example was splitting the ReturnRadio function into two. This function was used by both the RadioHomeScreen and the RadioInventoryScreen, but they each wanted to use it in different ways. Things like failure possibilities, result messages, and which screen to load upon completion, all differed between the two screens. Because they shared the function, it got messy. Splitting the function in two made it much cleaner.

Business design

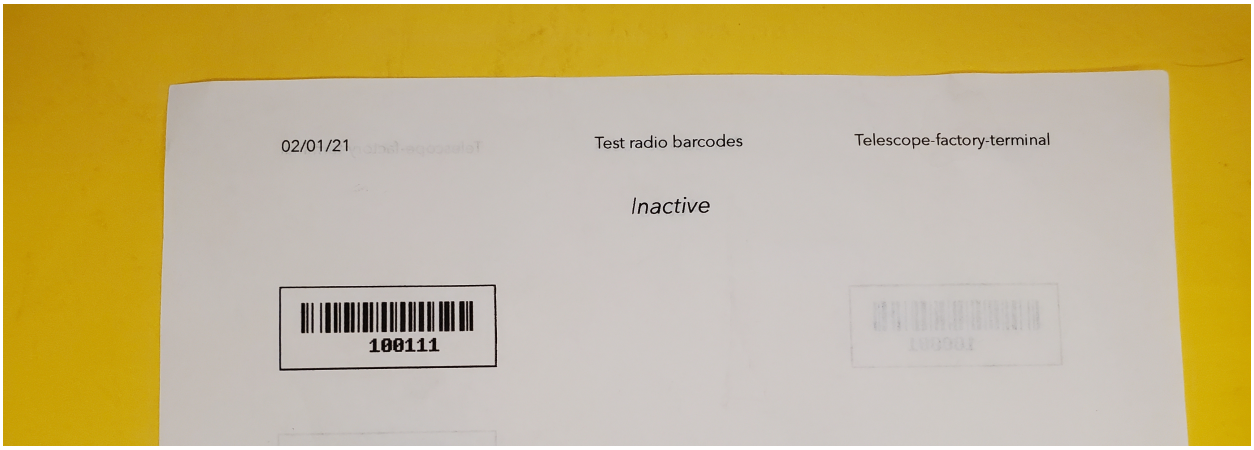
Before implementing the technicals of the project, there were several judgement-based design decisions to be made. As it turned out, conversations about these decisions continued throughout development as we considered the best options for real-world use. For example, one question was "Should users be allowed to clock out at the end of the day if they still have a radio checked out, given that all radios should be returned at the end of the shift?" We talked with the Telescope colleague in charge of hardware (including radios) and agreed that in theory the answer was 'no', but in reality it would be best to allow the clock-out. This was because if someone lost a radio and couldn't return it, preventing them from clocking out would not stop them from just going home anyway. The alternative would be to require them to ask a

supervisor to clock them out (and explain why), but we settled on requiring user confirmation to clock out while still owning a radio, and possibly (to be added in the future) sending an automatic email alert to the supervisor.

I was surprised by how many subtle yet important business questions came up like the one above. They required weighing the pros and cons, predicting possible misuse, and trying to understand how things work in the daily business operations.

Results

The radio system was completed with full functionality and no known bugs. I had the luxury of time to test the code (on both my desktop and the Elo touchscreens running on Raspberry Pi 3, which currently run the factory terminal code in production), and to address the lingering loose ends. Despite the relatively simple concept of the project it was surprising how many considerations, helper functions, and checks were necessary to do a thorough job. The completed project is designed to handle all cases in a known and predictable way.



Testing

The radio project has been merged into Telescope's master repository on Github, so it is now part of the most updated version of the factory terminal codebase running daily operations. One radio station has been set up on the factory floor, with the radio system live for employees to use. As seen in the above images, the user interface of the program is intuitive, so not too much user training was necessary. It was a successful launch with no problems.

Outcomes

This internship gave me the opportunity to work on a project from start to finish, and to experience the various stages of professional development. One lesson I took from it was the importance of being organized and systematic: keeping track of current bugs, having good version control, and completing one thing at a time to prevent forgotten loose ends. This applied to both the code and the business logic. To ensure the final product was good enough for use there were many questions, checks, and considerations––more than I had expected––to be handled. The process once again highlighted the importance of attention to detail.

Working with my business mentor and colleagues meant I got an understanding of the overall design of the system, including the design of the C# codebase and how that fit in with the other components of the technology stack. It was interesting to hear

about the reasons behind design decisions--the pros and cons of various options with respect to both technical and other considerations. My understanding has improved on both a technical and intuitive level.

The code I wrote for Telescope met the original specifications and is now being used in production. Getting to this point required a wide variety of strategies: upfront planning, trial and error when planning failed, asking for assistance when trial and error failed, persistence, and a change of strategy when needed, among many others. Over the course of the internship my skills have become better and more diverse.