PolyRents
Inventory Management System

By
Mitchel Davis

Computer Science Department
College of Engineering
California Polytechnic State University 2018

Date
Submitted: ______________________

Advisor: ______________________

Contents

1 Introduction

1.1 Problem Statement

The Orfalea College of Business Help Desk runs the facilities and resources of rooms 300 and 301 in the Orfalea College of Business at Cal Poly SLO. The help desk does not function as a business, but rather a resource to the College of Business. In addition to fielding and directing technology, software, and hardware inquiries, the help desk runs a rental system for the college's supply of laptops, chargers, calculators, and flash drives to students, professors, and staff. Thirteen students work as lab assistants and are qualified to lend out resources, and two of them work behind the desk during any given shift but typically, one student is managing rentals. This project details the implementation of a system - called PolyRents - to lend, track and aggregate resource rentals as well as prevent resource theft.

The current procedure is not secure, particularly in the laptop rental system. When a student requests to rent a laptop, their student ID is taken by the lab assistant and put in a file that matches the state ID number of the laptop that is checked out. The students last name is added to a Google Sheets document under the State ID number specific to the laptop. The sixteen laptops that are checked out far more often than other resources (ex. chargers, calculators and flash drives); when someone checks out a different resource, a sticky note detailing which resource they check out and the card with the sticky note is placed in a miscellaneous folder. The help desk has access to a desktop computer that could be equipped with a program to better keep track of rentals and ensure items are returned on time.

As previously mentioned, the current system only requires students to exchange their PolyCard for the rented item. The problem with this is that a PolyCard costs $5 and can easily be replaced; this means it carries little value to the student. Additionally, because the ink used to produce PolyCards is of poor quality, it is not a verifiable way to prove a students identity as the the photo can quickly fade enabling students to easily use another students PolyCard and in essence steal their identity. When the item at stake is a laptop costing over $1000, using a $5 ID as collateral is an inadequate way of tracking the technology.

1.1.1 Product Features

The system makes communication between students and employees much easier as the students'email and contact information is collected and stored by the system. The system also prevents students from being able to rent lab equipment without a Cal Poly issued PolyCard as a student must swipe their card before they can rent lab equipment. The system makes it easy to generate a report to show which resources are being used, and how frequently they are checked out. This allows the college to quantify the usefulness of the space, and get a better gauge for what students need and how to best satisfy these needs. In addition, student workers are able to run queries to determine which resources are currently allocated, to whom, and for how long.

1.2 Process Followed

The following steps were taken over Winter and Spring quarter of this year to implement this system.

1. Requirements Specification
2. System Design and Architecture
3. Development
4. Feedback
5. Repeat steps 3 and 4

Each of these steps are described in the following sections.

1.2.1 Requirements Specification

During the first half of Winter quarter when this project was started, requirements for the system were gathered from multiple stakeholders. Lab technicians were consulted as to ensure the product built would suit their daily workflow and to not introduce any extra work for them. The College of Business Director of Computing Resources, and manager of the lab technicians was consulted for technical requirements and specifications of the system. A representative from Cal Poly ITS Polycard Services was consulted to gather information about the type of data stored on the Polycard, how to extract it, and the requirements needed to be met in order to proper handle the information gathered. These requirements were recorded and are listed in Section 2 of this document.

1.2.2 System Design and Architecture

Once the necessary requirements were gathered, they were used during the design phase of this project. The current system does not satisfy all of the requirements gathered, but design decisions were made during the architecture phase to allow for easy implementation of these requirements.

1.2.3 Development

Development of this project started during week five of Winter Quarter and was carried out over 6 two week sprints spanning both Winter and Spring Quarter. The requirements gathered were converted into user stories and tasks and were logged in a GitHub Issue repository. At any given time two sprints were planned out using GitHub's Project tool and the work for each sprint was taken directly from the Issue backlog mentioned earlier.

1.2.4 Feedback

work carried out during each sprint was then reported and demoed to Professor da Silva. After each sprint, the system handed over to Lab technicians to validate the functionality of features implemented, find bugs, and gather new requirements. Bugs, feature requests, and UX enhancements were logged and added to the GitHub Issue Repository mentioned above and fixed/implemented in subsequent sprint.

Links to the above mentioned Github Repository, Issue Repository, and Project are listed below:

$$https://github.com/mdavis60/Polyrents$$

$$https://github.com/mdavis60/PolyRents/issues$$

$$https://github.com/mdavis60/PolyRents/projects/1$$

1.3 Paper Outline

The format of the rest of the paper will proceed as follows:

1. Requirements and Features
2. System Architecture
3. UI Screen Shots
4. Retrospective
5. Conclusion

2 Requirements and Features

In this section the functional and business requirements gathered during the first half of Winter Quarter and during development are listed. For simplicity, only the most important requirements are listed.

2.1 Functional Requirements

In Table 1, the functional requirements gathered at the beginning of the project are listed. It is important to note that the requirements listed are the only the core requirements that cover the major features of PolyRents. For a full list please refer to the Issue Repository mentioned in section 1.2.

ID	Functional Requirement
FR-01	The system must allow the user to swipe their Polycard so they can register and checkout lab equipment.
FR-02	The system must store information pertaining to lab equipment so it can be tracked and queried on for data analyses.
FR-03	The system must store information pertaining to students so they do not have to register in the system more than once.
FR-04	The system must store information pertaining to rental transactions so the lab equipment location is known at all times.
FR-05	The system must allow a user to checkout multiple pieces of lab equipment at once so that faculty members can use them for their classes.
FR-06	The system must interface with the mySQL database so information about the computers used by the system is accessible.
FR-07	The system must implement and use a Microsoft Access Database as its main database so that information pertaining to rental history can be accessed and queried easily.
FR-07	The system must implement and use a Microsoft Access Database as its main database so that information pertaining to rental history can be accessed and queried easily.

Table 1: List of Functional Requirements

2.2 Business Rules

In the system's current state, no requirements must be met in order to comply with Cal Poly's Information Classification and Handling Standards. However, as the system develops and matures, some of these standards will have to be satisfied, specifically the standards outlined for level two information. For a full list of these standards please visit:

$$https://security.calpoly.edu/content/policies/standards/classification/index$$

3 Architecture Overview

In this section, the architecture of the system developed during the past two quarters is provided. The diagrams shown depicted both high and low level design of the system.

3.1 Components

Figure 1 shows the high level design of the application. WPF applications follow the Model View ViewModel (MVVM) design patter. These layers can be show in the in their corresponding components.

The Model Component is located within the Data Access Layer which contains the Computing Resources Data Set Component as well. This layer is responsible for interfacing directly with the Access Database through the corresponding AccessDB driver. The Access database is located in the Data Layer along with the MySQL Server. The primary method of communication between the Access Database and the MySQL database is the MySQL ODBC Driver.

The View and ViewModel Components are located within the Screens Component. The reason for this is because for each WPF Screen, there is a corresponding .xaml and .cs file corresponding the the View and ViewModel components respectively. Typically each view has a model that acts as the data binding and the ViewModel responds to UI events generated by the View and updates the model respectively. These changes are first persisted through the ComputingResourcesDatSet in the Data Access Layer before they are propigated back to the Screens layer where the changes are show in the View.

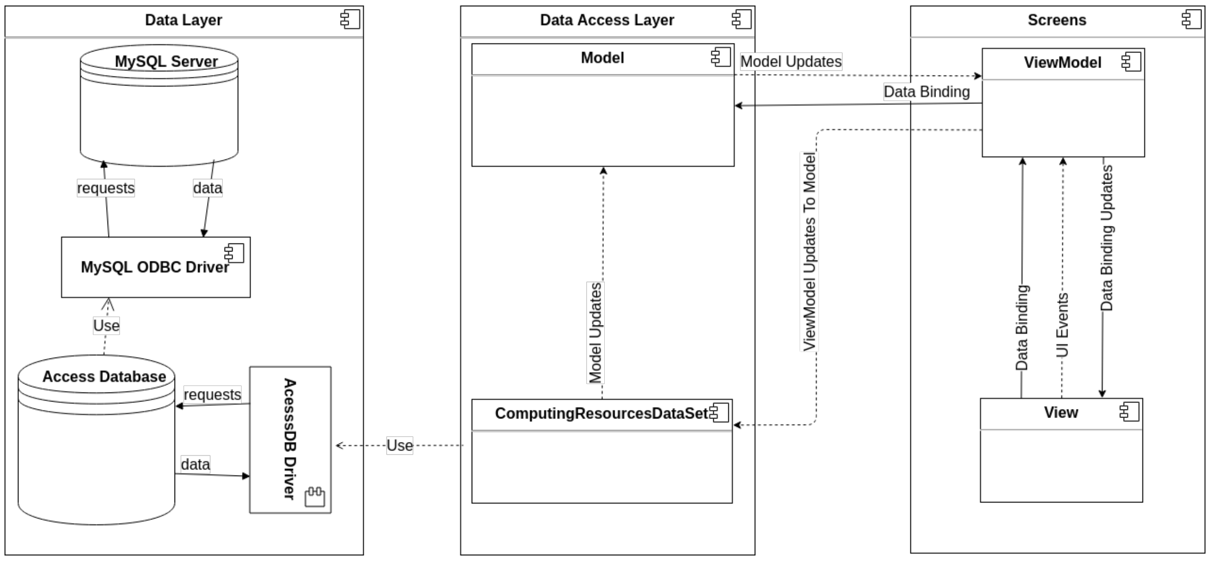


Figure 1: Component Diagram

3.2 Entity Relation Diagram

Figure 2 depicts the database models, their relationship to each other, and where in the system they could be found. The computers table shown on the left was the only table from the mySQL database that was used in this system. The remaining entities resided in the Access Database. A join table was needed to link resources used by the System primarily in the Access Database to the corresponding computer information that resided in the mySQL Database. This figure also shows how the system was modeled.

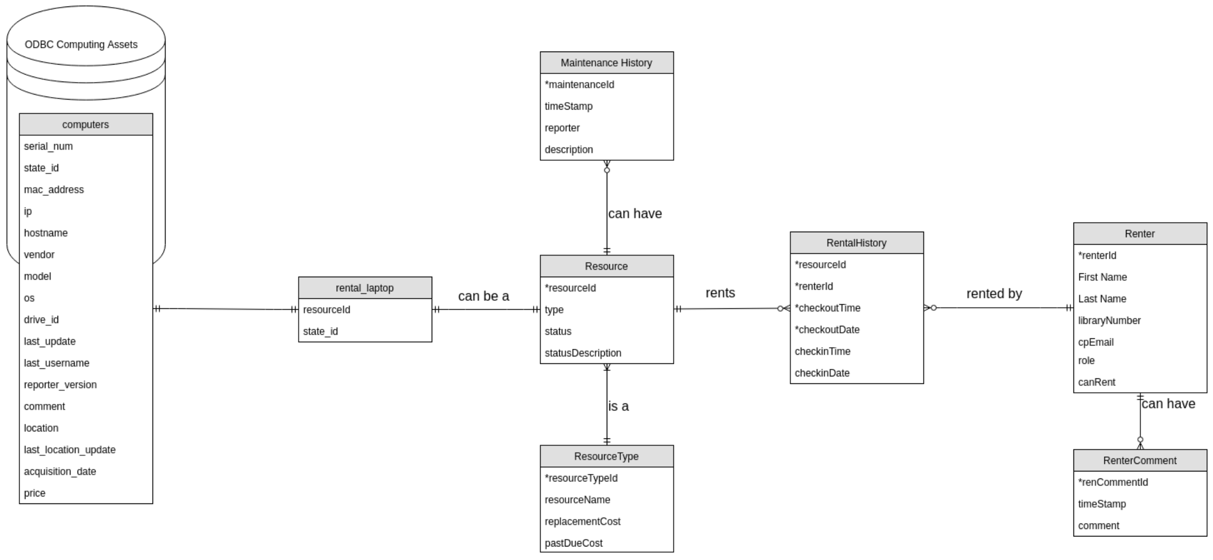


Figure 2: Entity Relation Diagram

3.3 Class Diagram

Figures 3 and 4 show the a detailed model of the Data Access Layer portion of the WPF application. When using a data source in the Visual Studio IDE, the IDE generates a TableAdapter class for each table in your database. These classes are depicted in the upper half of figure 3. Each one of these classes has a corresponding Converter Class which converts the data fetch from the Access Data base into its corresponding C sharp model class; these classes are depicted in the lower half of figure 3. For readability, the diagram above depicts just four of the seven auto generated Table Adapter and user defined DAO interfaces and converter classes.

By leveraging a feature of the C sharp language, called partial classes, I was able to implement the user defined DAO interfaces, depicted in figure 4, directly inside the auto generated code. This not only reduced the complexity and increased the cohesion of my data access layer, but removed the dependency on the Data Access Layer from the Presenter Layer of the Application, which expects my DAO interfaces and not the auto generated TableAdapter classes.

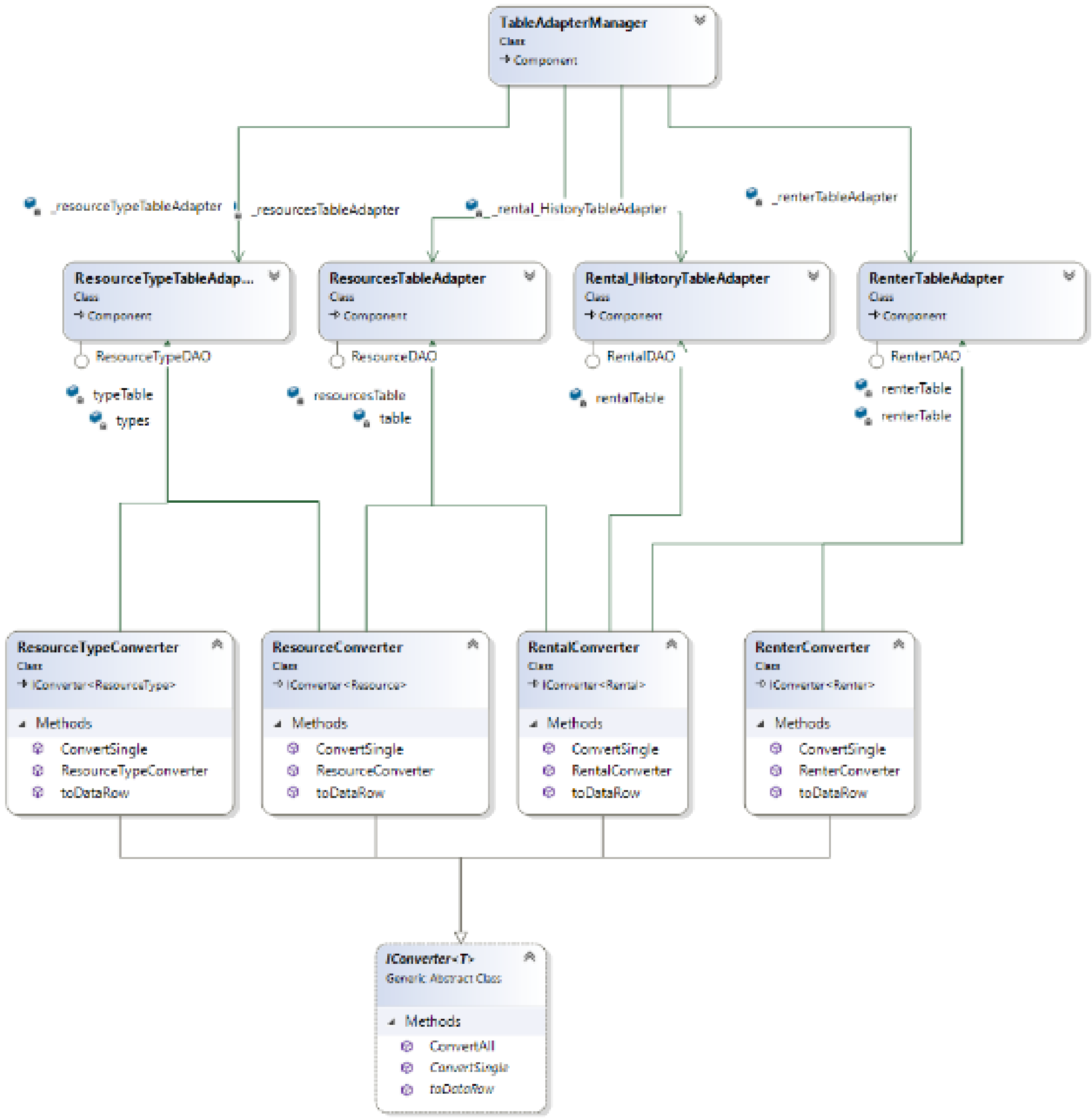


Figure 3: Data Access Layer Class Diagram1

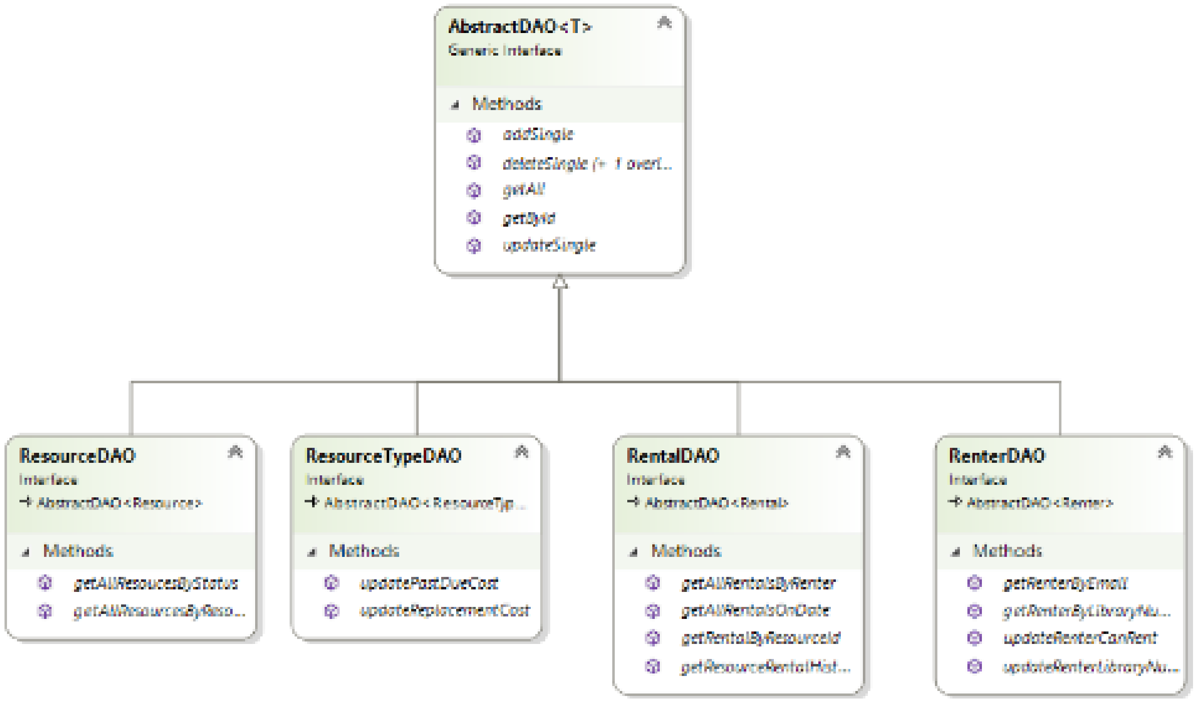


Figure 4: Data Access Layer Class Diagram2

4 UI Screen Shots

In this sections screen shots of the application are shown. To simplify the user interaction with the system, only three types of screens were made.

1. Home Screen
2. Add or edit screens
3. Manage screens

4.1 Home Screen

The Home screen, shown in figure 5, displays a list of pages the user can choose to navigate directly to. These options were chosen and ordered based on the frequency of the use cases.

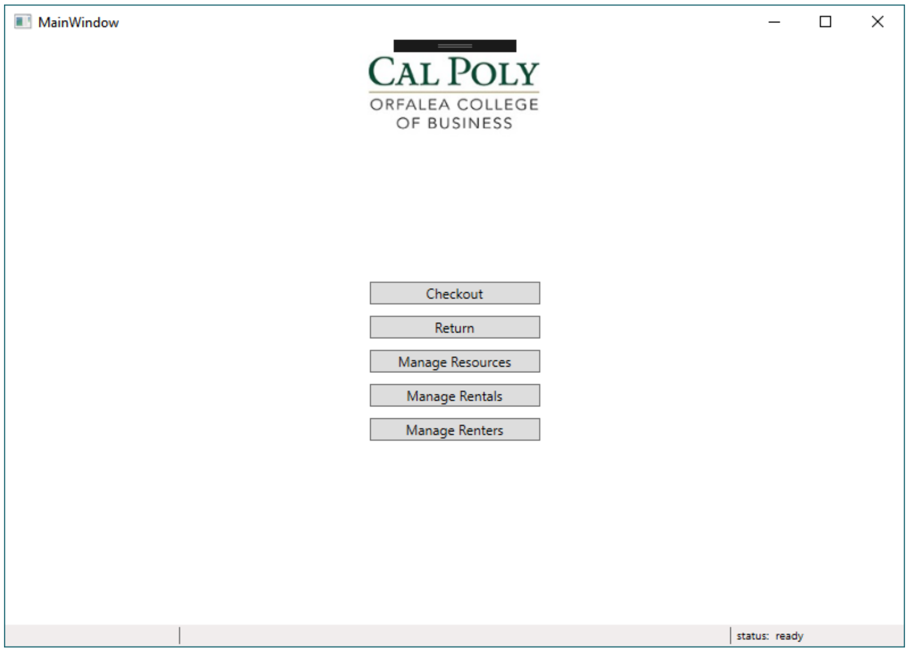
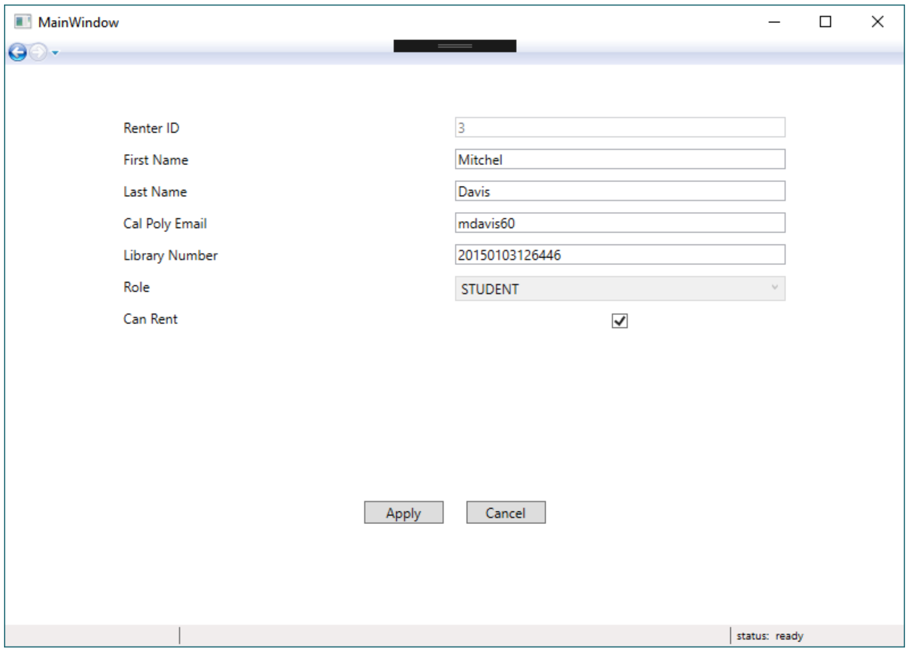


Figure 5: Home Screen

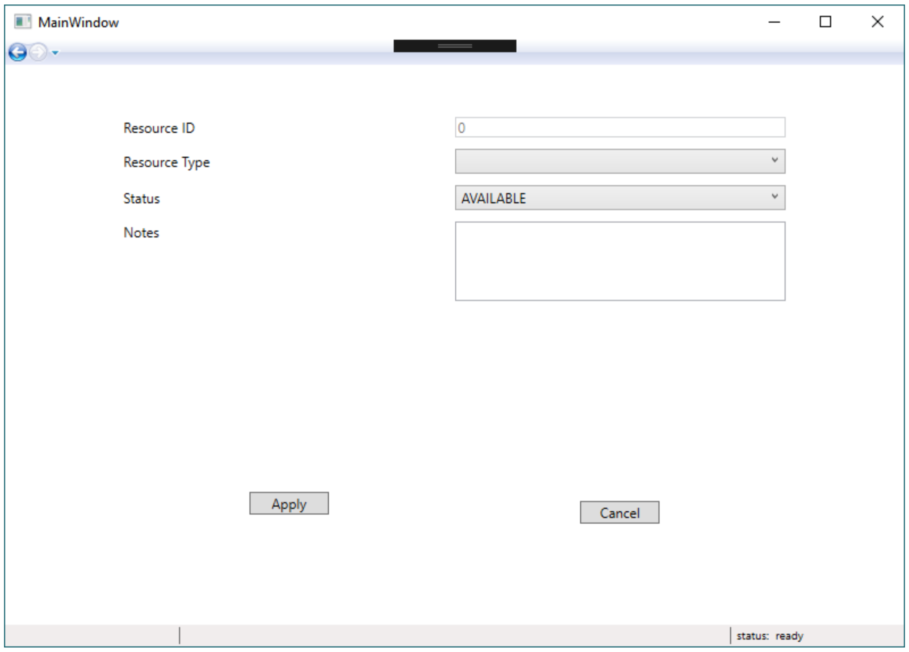
4.2 Add or Edit Screens

The Add or Edit Screens, shown in figures 6-9, provide lab technicians the ability to create and update information in the system. With the exception of the checkout and return screens, the Add and Edit screens were made to mirror each other as closely as possible in order to simplify user interaction.



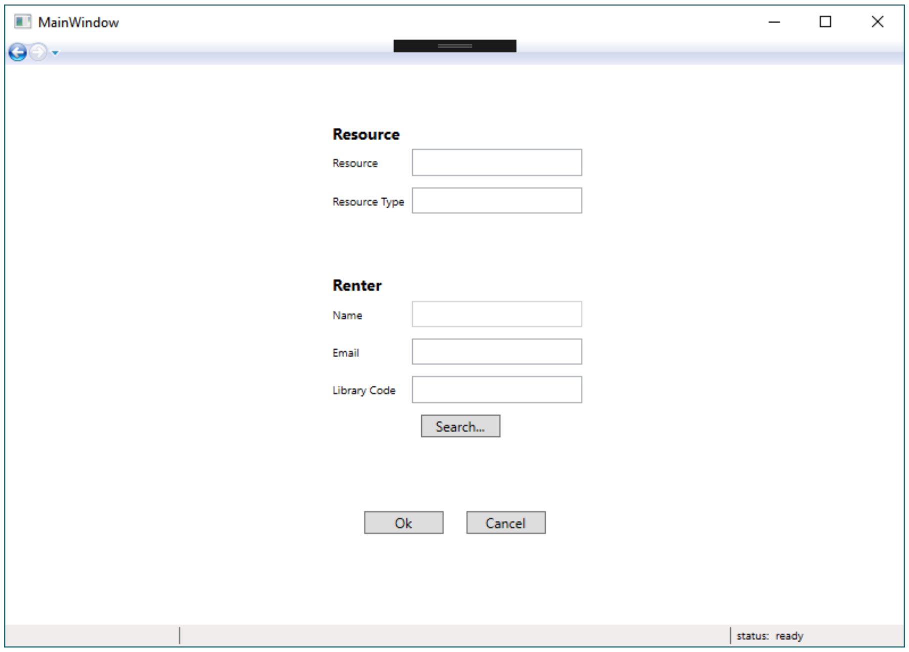
Renter ID	3
First Name	Mitchel
Last Name	Davis
Cal Poly Email	mdavis60
Library Number	20150103126446
Role	STUDENT
Can Rent	✓

Figure 6: Edit Renter


MainWindow

Resource ID	0
Resource Type	
Status	AVAILABLE
Notes	

Figure 7: New Resource


MainWindow

Resource

Resource	
Resource Type	

Renter

Name	
Email	
Library Code	

Figure 8: Checkout

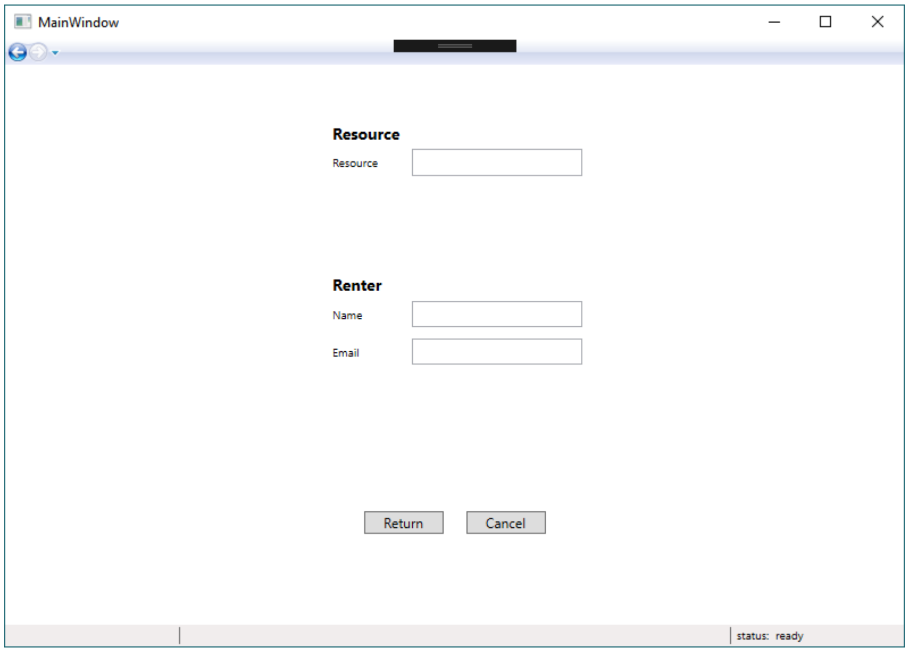
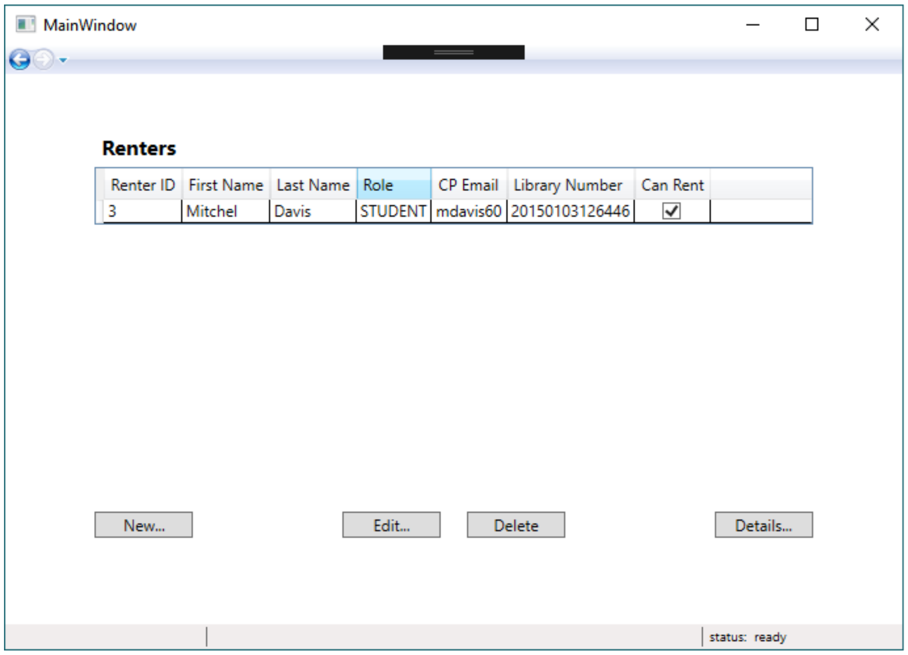


Figure 9: Return

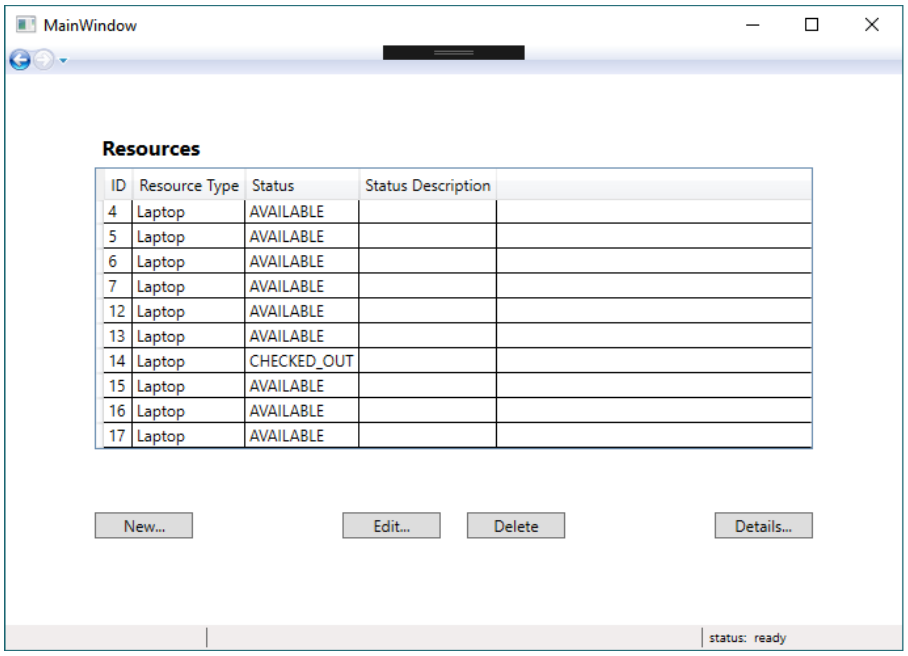
4.3 Manage Screens

The Manage screens, shown in figures 10-12, provide lab technicians the ability to view a detailed list of all resources, renters, and rentals. These pages were designed to be as similar to each other as possible and enable the lab technician to quickly



Renter ID	First Name	Last Name	Role	CP Email	Library Number	Can Rent
3	Mitchel	Davis	STUDENT	mdavis60	20150103126446	✓

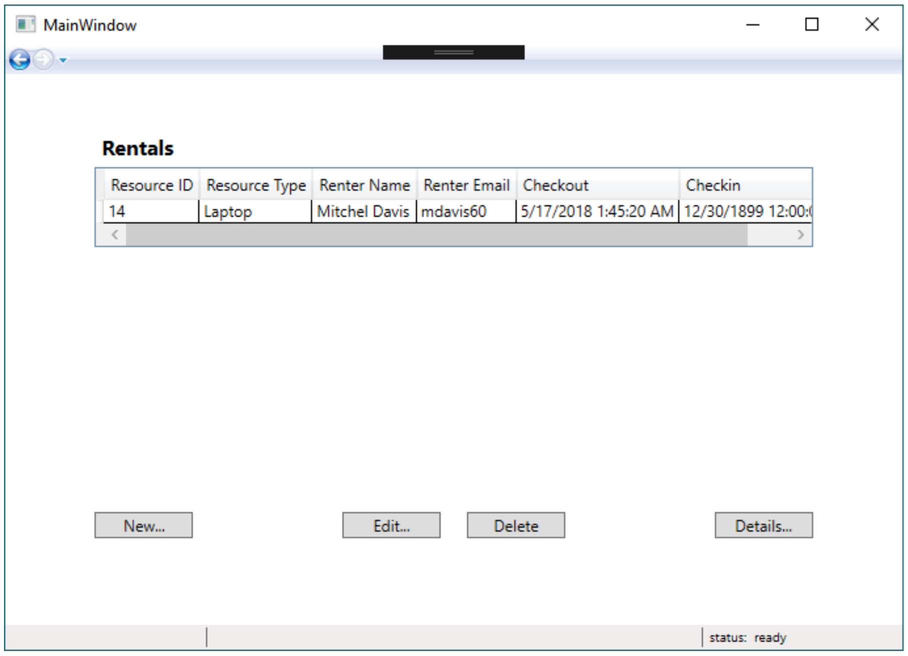
Figure 10: Manage Renters



Resources

ID	Resource Type	Status	Status Description
4	Laptop	AVAILABLE	
5	Laptop	AVAILABLE	
6	Laptop	AVAILABLE	
7	Laptop	AVAILABLE	
12	Laptop	AVAILABLE	
13	Laptop	AVAILABLE	
14	Laptop	CHECKED_OUT	
15	Laptop	AVAILABLE	
16	Laptop	AVAILABLE	
17	Laptop	AVAILABLE	

Figure 11: Manage Resources



Rentals

Resource ID	Resource Type	Renter Name	Renter Email	Checkout	Checkin
14	Laptop	Mitchel Davis	mdavis60	5/17/2018 1:45:20 AM	12/30/1899 12:00:

Figure 12: Manage Rentals

5 Project Retrospective

5.1 Lessons Learned

Throughout the course of this project, I pushed myself out of my comfort zone in nearly every aspect of this project. I chose to use a language (C#) and framework (.NET), and IDE (Visual Studio) is was not familiar with. I chose a project that interface with a piece of hardware (card reader) and to connect to a database that was itself connected to another database. This combination of challenges made development slow. While researching solutions to one problem, I would find more elegant solutions to old problems that made the design of the application simpler.

As a result I would often times find myself rewriting the same code three or more times, each time making it slightly less complex and more readable. As a result I not only learned a lot about C# and the .NET framework, but the code I produced for this project has been some of the cleanest code I have ever written. After completing this project, I am more confident in my design and development skills.

That being said, the time spent rewriting the same code over and over was time wasted. More functionality could have been added to the system if I had chosen to not change my previous code. However, keeping in mind I will be handing this project of to other underclassmen computer science and software engineering majors that work at the OCOB Helpdesk, I feel taking the time to make sure the code that was written was of high quality will ultimately lead to greater future success for this project.

6 Conclusion

The goal of this project was to create a system that would replace the manual tracking of computer rentals at the OCOB Helpdesk to mitigate the risk involved with loaning laptops to students and faculty, and to provide a framework for future development to take place so the tool built upon and used for other use cases. PolyRents proved to be a viable product. During the feedback gathered with potential users of the system, constructive criticisms and a general approval of the system was shared. Through continuous user testing and iterative development, the system created satisfies the original goal.

6.1 Future Work

In order for this system to continue to work, regular maintenance and bug fixes will need to be performed. As it currently stands the system has various minor bugs with known workarounds, but in the case a novice user starts to use the system, loss of data can occur. Fortunately, most of the effort that was expended during this project was spent writing readable code and refactoring internal data structures to reduce complexity and increase cohesion between classes so that another developer could pick up this project and continue development with out too much trouble.