Silicon Tracking Solutions

Brett Hayes, Joseph Cronise, and Dylan Camus
CS463 / Spring 2016 / Group 03
Final Report

Abstract

This document contains the work of Brett Hayes, Joseph Cronise, and Dylan Camus pertaining to the Silicon Tracking project. This project is a web-based inventory system used by the Performance, Measurement, and Analysis team at Intel in Chandler, AZ. The inventory system we have designed tracks the information and status of CPU silicon used by the team and is made to help them to know where their high value inventory is at all times, as well as saving the team time and effort towards obtaining the items they need to get their work done. We have designed for them a system such that every time an engineer within their lab needs to use an item, they will need to check out the item through our system and check it back in when they are done. This ensures that highly sensitive materials don't go missing without a log of who has had ownership of the item, as well as letting others know who currently is in ownership of the item.

CONTENTS

I Introduction 3
I-A Who Requested it? . . . 3
I-B Why Was It Requested? . . . 3
I-C What Is Its Importance? . . . 3
I-D Who Was/Were Your Client(s)? . . . 3
I-E Who Are the Members of Your Team? . . . 3
I-F What Were Their Roles? . . . 3
I-G What Was the Role of the Client(s)? (I.e., Did They Supervise Only, or Did They Participate in Doing Development) . . . 3

II The Original Requirements Document 3
II-A Introduction . . . 3
II-A1 About This Document . . . 3
II-A2 Definitions . . . 4
II-A3 Current System . . . 4
II-B The Client would like to... . . . 4
II-B1 Inventory . . . 4
II-B2 Security . . . 5
II-B3 Database . . . 5
II-B4 Services . . . 6
II-C The Interfaces... . . . 6
II-C1 Main Interface . . . 6
II-C2 Shopping Cart Interface . . . 6
II-C3 Scrapping Interface . . . 6
II-C4 Reporting Interface . . . 7
II-D The Expo Should Look Like... . . . 7
II-D1 Web Application . . . 7
II-E Appendix . . . 7
II-E1 Multi-Level Filtering . . . 7
II-E2 Gantt Chart . . . 7

III Changes Since the Original Client Requirements Document 9
III-3 The New Gantt Chart . . . 10

IV The Original Design Document 12

V Changes Since the Original Design Document 27

VI The Original Technology Document 27
VI-A What We Are Trying to Accomplish . . . 27
VI-B Cameras . . . 27
VI-B1 Standard CCTV camera . . . 27
VI-B2 Intel RealSense Camera(F200) . . . 28
VI-B3 Kinect Sensor . . . 28
VI-B4 Selection Criteria . . . 28
VI-C RFID Reader . . . 28
VI-C1 1126 Desktop UHF RFID Reader . . . 28
VI-C2 ID CPR02.10-AD/-B RFID Card Reader . . . 28
VI-C3 pcProx82 Series . . . 28
VI-C4 Selection Criteria . . . 28
VI-D Database Management System . . . 28
VI-D1 MySQL . . . 28
VI-D2 Microsoft SQL Server Express . . . 29
VI-D3 Oracle Database Express . . . 29
VI-D4 Selection Criteria . . . 29
VI-E Web Application . . . 29

VI-E1 Node.js / Angular.js . . . 29
VI-E2 Visual Studio Community . . . 29
VI-E3 Vanilla HTML/CSS/Javascript with PHP . . . 29
VI-E4 Selection Criteria . . . 29

VII Changes Since the Original Technology Document 29

VIII Weekly Blog Posts 30
VIII-A Welcome to my blog! by Cronise, Joseph Eric . . . 30
VIII-B Weekly Update 10/23/2015 by Cronise, Joseph Eric . . . 30
VIII-C Weekly Update 10/30/2015 by Camus, Dylan Michael . . . 30
VIII-D Weekly Update 11/6/2015 by Camus, Dylan Michael . . . 30
VIII-E Weekly Update 11/13/2015 by Camus, Dylan Michael . . . 30
VIII-F Weekly Update 11/27/2015 by Hayes, Brett . . . 30
VIII-G Weekly Update 12/5/2015 by Hayes, Brett . . . 31
VIII-H Weekly Update 1/16 by Hayes, Brett . . . 31
VIII-I Weekly Update 1/29 by Hayes, Brett . . . 31
VIII-J Weekly Blog Post 2/5 by Hayes, Brett . . . 32
VIII-K Weekly Update 2/12 by Camus, Dylan Michael . . . 32
VIII-L Weekly Blog Post 2/19 by Hayes, Brett . . . 32
VIII-M Weekly Blog Post 2/26 by Hayes, Brett . . . 32
VIII-N Weekly Blog Post 3/4 by Camus, Dylan Michael . . . 33
VIII-O Weekly Blog Post 3/11 by Hayes, Brett . . . 33
VIII-P Weekly Blog Post 4/8 by Hayes, Brett . . . 33
VIII-Q Weekly Blog Post 4/15 by Hayes, Brett . . . 33
VIII-R Weekly Blog Post 5/9 by Hayes, Brett . . . 34
VIII-S Weekly Blog Post 5/15 by Hayes, Brett . . . 34
VIII-T Weekly Blog Post 5/27 by Hayes, Brett . . . 34

IX Project Documentation 35
IX-A How Does Your Project Work? . . . 35
IX-B How Does One Install the Project? . . . 36
IX-C How Does One Run the Program? . . . 37
IX-D Are There Any Special Hardware, OS, or Runtime Requirements to Run Your Project? . . . 38

X How Did You Learn New Technology? 38
X-A What websites were helpful? (Listed in order of helpfulness.) . . . 38
X-B What, If Any, Reference Books Really Helped? . . . 39
X-C Were There Any People on Campus That Were Really Helpful? . . . 39

XI What Did You Learn From This? (Dylan Camus) 39
XI-A What Technical Information Did You Learn? . . . 39
XI-B What Non-Technical Information Did You Learn? . . . 39
XI-C What Have You Learned About Project Work? . . . 39
XI-D What Have You Learned About Project Management? . . . 39
XI-E What Have You Learned About Working in Teams? . . . 39
XI-F If You Could Do It All over, What Would You Do Differently? . . . 39

XII What Did You Learn From This? (Brett Hayes) 40
XII-A What Technical Information Did You Learn? . . . 40
XII-B What Non-Technical Information Did You Learn? . . . 40
XII-C What Have You Learned About Project Work? . . . 40
XII-D What Have You Learned About Project Management? . . . 40
XII-E What Have You Learned About Working in Teams? . . . 40
XII-F If You Could Do It All over, What Would You Do Differently? . . . 40

XIII What Did You Learn From This? (Joseph Cronise) 40
XIII-A What Technical Information Did You Learn? 40
XIII-B What Non-Technical Information Did You Learn? 40
XIII-C What Have You Learned About Project Work? 41
XIII-D What Have You Learned About Project Management? 41
XIII-E What Have You Learned About Working in Teams? 41
XIII-F If You Could Do It All over, What Would You Do Differently? 41

XIV Appendix 1: Essential Code Listings 41

XV Appendix 2: Additional Photos 67

I. Introduction

A. Who Requested it?

This project was requested by our client Scott, along with his team at Intel.

B. Why Was It Requested?

Scott's team needs to keep inventory of all of their CPU silicon in their lab. These silicon pieces have no real identifiable marks other than a serial number. The members of Scott's lab needed an easy way to retrieve information about the silicon. The other reason for this inventory system is to know who has which silicon. Each piece of silicon is considered very valuable to Intel, so they need a way to know who has taken silicon in order to do testing. They also want to know who has which silicon for their own team's sake. They like to know if a certain piece of silicon is being used by another team member, and to know as soon as their teammate is done with it.

C. What Is Its Importance?

It is important for the team members to have an easy-to-use system that helps them do their work more efficiently than they could without it. It is meant to save them time, and get to the important parts of their work. This system is also important in preventing the loss of silicon, which could be very costly.

D. Who Was/Were Your Client(s)?

Scott Oehrlein

E. Who Are the Members of Your Team?

Brett Hayes, Joseph Cronise, and Dylan Camus

F. What Were Their Roles?

None of the members took on a specific role for the project. There were many tasks to the project, so everyone had to take on multiple roles, with much overlap between roles. The description of the roles provided will be about which member was the primary caretaker for that part of the project.

Brett was responsible for the server-side implementation. He created the basic structure and maintained a lot of the server-side code. He was also responsible for a lot of the dynamic code running on the client. Brett also set-up a task list each term, to make sure the team was on track throughout the project.

Dylan was in charge of the database. He worked on the design structure, and created many of the stored procedures used in the current system. He was also responsible for the emailing system, which included sending immediate emails and scheduled emails.

Joseph came up with the original design of the website, and was responsible for the research involved in facial recognition.

G. What Was the Role of the Client(s)? (I.e., Did They Supervise Only, or Did They Participate in Doing Development)

The client played a supervisor role, and he had his team manage the setup of our project into their environment. We tried to make sure that our project was simple to set up, and would work in different environments, since we do not have any knowledge of their environment. Our client's team also took care of most of the security of the project, since that is also knowledge-based and they are the only ones with access to that knowledge.

II. The Original Requirements Document

This is the Requirements Document we wrote at the beginning of the project. It has been reformatted to keep consistent styling with this document, but the content of the document has not been modified. We have also indexed the requirements in this document for easy reference. The original document in its original formatting is included on the usb storage drive.

A. Introduction

1) About This Document: This is the requirements document for the Silicon Tracker project. Included in the following sections are lists of user stories organized by categories. Each paragraph contains a feature to be added to the web application. They are all organized into their respective categories and are set up as individual tasks to be checked off as they are completed.

2) Definitions: The kiosk mentioned in a few user stories is a nickname that the Intel PMA Labs has given for their check in/check out system. They have a computer with a QR scanner that sits on top of a large locker with all the items they store.

Although one of the main items being tracked will be silicon, we use the generic term item to specify that any part needed, not just silicon, can be inventoried by our system.

3) Current System: The current system Intel uses is inadequate for their needs. The items have to be checked in or out one at a time. This means the user has to enter their credentials for every item that goes through the system. They are wanting a more user-friendly web application where they can add items to their shopping cart and when they are ready, go to the kiosk and check in and out all their items in a single transaction.

B. The Client would like to...

1) Inventory:

Req 1: Queue an item that the user wants but is not checked in. If a person needs this item and it is currently checked out, they can be put on a queue for that item.

Req 2: Give a reason and a priority when queueing an item. This will create a priority queue for the users. If someone needs an item and it is important, they can give a reason why.

Req 3: When an item gets checked in and there were people waitlisting on the item, the people who have waitlisted will be notified with everyone's priority and reason for needing the item.

Req 4: Check out an item that is checked into the system. If nobody else has the item reserved, they can check out the item immediately.

Req 5: Check in an item that was checked out of the system. The user returns the item so other people can check out the item.

Req 6: Check in multiple items in a single transaction. The current system doesn't allow this feature. This would let the user check in multiple items without giving their credentials for every item.

Req 7: Check out multiple items in a single transaction. The current system doesn't allow this feature. This would let the user check out multiple items without giving their credentials for every item.

Req 8: Check in and check out multiple items in the same transaction. Along with the above two stories, the user should be able to both check in and out and only give their credentials once.

Req 9: Enter new items into the system. There should be a form that a user can fill out. They will scan the item to insert the item's ID.

Req 10: Retire old items in the system. They need to logically delete (scrap) items when an item is no longer being used. There should be a textbox to optionally give a reason for the item being retired.

Req 11: Be able to scan an item when checking in and have that automatically update the database. The user should just be able to log in, and then scan the item. The status of the item should be updated based on the item being scanned. This should happen when the user has the item currently checked out.

Req 12: Be able to scan an item when checking out have that automatically update the database. The user should just be able to log in, and then scan the item. The status of the item should be updated based on the item being scanned. This should happen when the user has the item.

Req 13: Be able to scan an item that is currently checked in to see who has the item reserved next. This will show the next person in line for the item.

Req 14: Be able to scan an item that is currently checked out to see who has the item checked out. For example, if there is an item that someone left out and nobody knows who has it checked out, the item can be scanned to see who it currently belongs to.

Workflow of the system:

Req 15: The initial screen is a welcome screen. This is where the user can scan their badge, use facial recognition, or enter a username/password to login. After proper authorization, the user is presented with the shopping cart interface.

Req 16: The user can now start scanning items.

Req 17: If the item is scanned and the item is checked in, it will now be checked out by the user. If the item was already checked out, information about who currently has the item checked out will be displayed on the screen.

Req 18: If the current user has an item checked out, they can scan the item and the item will be checked in.

Req 19: When the user is done, they will press a confirm button and will be shown a list of items they have checked in/out. The user can then proceed to commit their transaction to the database.

2) Security:

Req 20: Log into the application with a username and password.

Req 21: User settings and permissions will be maintained in the project database. Authenticate user credentials using the Active Directory server already set up at Intel. They already have a system in place and the users will have to sign in with their current credentials.

Req 22: Authenticate facial recognition using the project database. The data stored for facial recognition will have to be in the project database, since we cannot modify the Active Directory server.

Req 23: When setting up facial recognition for a user, authenticate user with username and password, or with RFID tag first. This is for security reasons, so other people won't try to scan their face on someone else's profile. When checking in or out item(s), the user should be able to authenticate with their current RFID tags using an RFID reader.

Req 24: When checking in or out item(s), the user should be able to authenticate with facial recognition.

Req 25: When checking in or out item(s), the user should be able to authenticate with username and password.

Req 26: Those with elevated permissions can retire old items from the system. This will be a logical delete, not an actual delete from the database.

Req 27: Those with elevated permissions can modify certain fields of the items.

Req 28: Have any communication between the web server and the client will be encrypted through SSL. The information about the items are all confidential, and they need to stay that way.

Req 29: Admins will have the ability to enable/disable any of the three authentication methods (username/password, RFID tag, or facial recognition). If for whatever reason the admin does not want one of these methods available, they can disable that method of authentication.

3) Database:

Req 30: Have new items be inserted into the project database.

Req 31: Have old items continue to be stored in the database after they are retired (logically deleted). Certain users still need the information about items even if they are no longer active.

Req 32: Have a record of who checked in which items, and have that stored in the database. This creates an audit trail for all the items.

Req 33: Have a record of who checked out which items, and have that stored in the database. This creates an audit trail for all the items.

Req 34: Have a timestamp for every time an item is inserted, updated, or retired. This is also for auditing reasons.

4) Services:

Req 35: Have an e-mail sent out when an item is queued to any person that has the specific item checked out.

Req 36: Send out periodic schedulable reports via e-mail to each user based on the items they checked in and out, and have on queue.

Req 37: Have users be notified via e-mail when they check in an item. This should happen for every transaction, so the user doesn't get a separate e-mail for every item.

Req 38: Have users be notified via e-mail when they check out an item. This should happen for every transaction, so the user doesn't get a separate e-mail for every item.

Req 39: Have users be notified when an item is now considered retired and they have the item currently checked out. They will need the e-mail so they know to return it immediately.

C. The Interfaces...

1) Main Interface:

Req 40: Will be a touch interface without need of keyboard. The keyboard will still be available for certain functions, such as entering a new item. The idea is that the user doesn't have to use the keyboard for check-in and check-out transactions.

Req 41: Will be able to check in items from this interface. This should not require a keyboard. The user should simply login to the system and scan the item(s) to check them in.

Req 42: Will be able to check out items from this interface. This should not require a keyboard. If they have the item ready for check out, then they should simply login to the system and scan the item(s) to check them out.

Req 43: Will be able to scan an item and display the information about that item (i.e. in the case of silicon, Core Count, LLC Size, Frequency, Status (Checked in or Checked out), etc.)

2) Shopping Cart Interface:

Req 44: Will be able to check in items from this interface. If they are at the kiosk they should have a way to check in items using this interface.

Req 45: Will be able to check out items from this interface.

Req 46: Will be able to add new items to the system from this interface. When a new item comes in, they will need to enter information about the item, and then scan it so they have the serial number recorded.

Req 47: Will be able to retire items from circulation (with elevated privileges). The items will be scrapped, but still in the database for record keeping.

Req 48: Will be able to reuse information of items when the same item is being added to the inventory system. They will have multiples of the same item. That item should just be scanned and linked with the other items in the system.

Req 49: Will be able to search for items based on certain fields in the database. The user should be able to filter their search results based on the item attributes.

Req 50: Will have multi-level filtering on searches. For more information, see Appendix Multi-Level Filtering.

3) Scrapping Interface:

Req 51: If a user has the right elevated privileges, they can be taken to a scrapping interface. This interface will be similar to the shopping cart interface. The difference will lie in the fact that any item the user scans will be scrapped instead of checked in/out. They will confirm and commit their transaction, just like the shopping cart interface.

4) Reporting Interface:

Req 52: Will be able to view any item in circulation. This should be in the form of a user-friendly table with pagination.

Req 53: Will be able to view any item that is retired. This should also be in the form of a user-friendly table with pagination.

Req 54: Will be able to see what items a certain user has checked out. They should be able to click on a user, and it will give a list of items in a user-friendly table with pagination.

Req 55: Will be able to search for any user who checked out a certain item. This will display as a table with the users who have one or more of these items.

Req 56: Will have multi-level filtering on searches. For more information, see Appendix A Multi-Level Filtering.

D. The Expo Should Look Like...

1) Web Application:

Req 57: A Fully functioning version will be running during the event. This will be a system that the general audience can view and play with.

Req 58: Anybody can have their faces entered into the database. This will show how the facial recognition works.

Req 59: Anybody can scan sample RFID cards to check in or out items. We will have sample items for them to scan, and they can use fake RFID badges to simulate the experience.

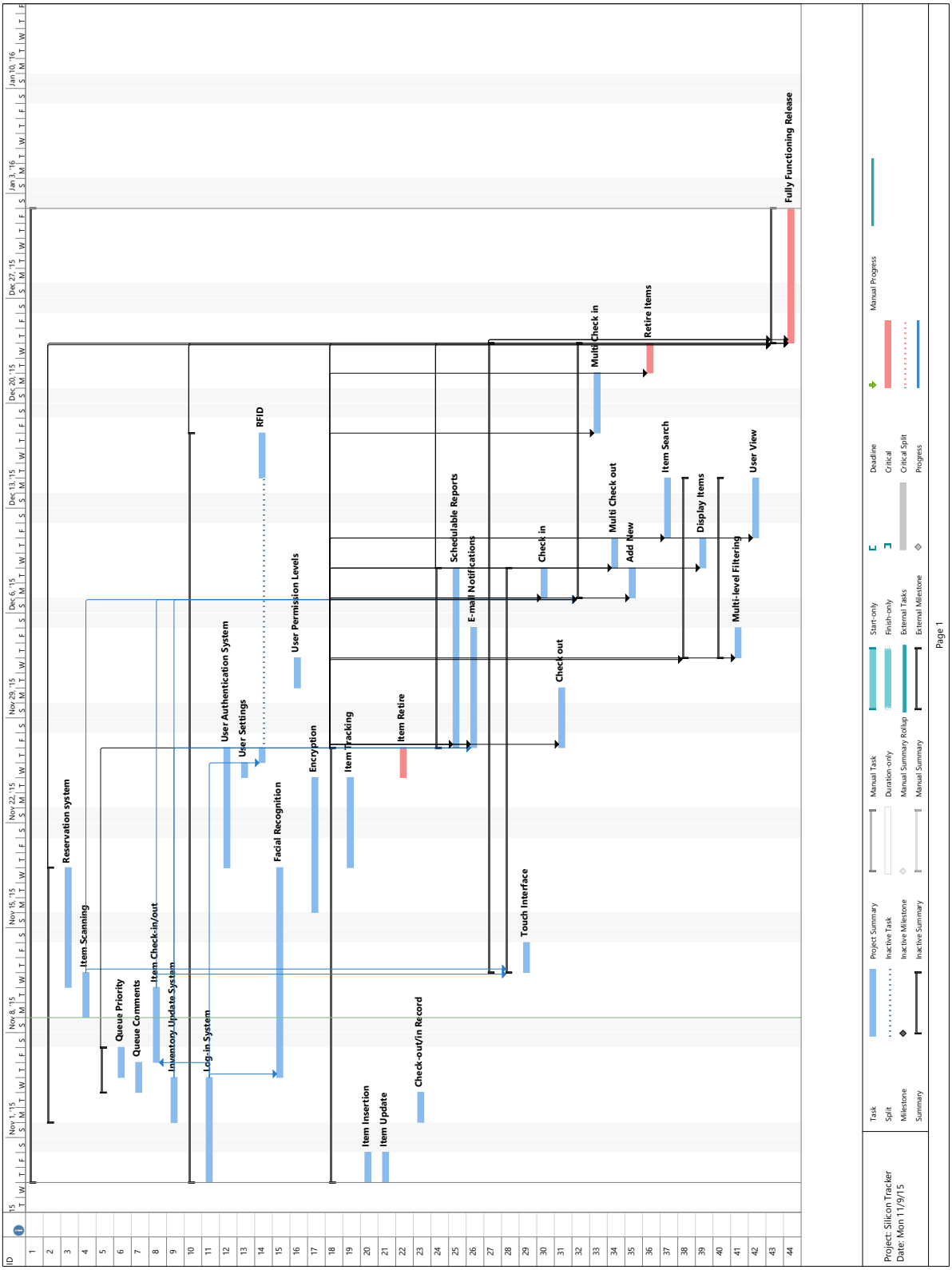
Req 60: Anybody can scroll through the different interfaces. They will have admin privileges so they can see all the features of the application.

Req 61: Anybody can check in or out items into the system. There will be a set of sample items people can scan so they can see how the system works.

E. Appendix

1) Multi-Level Filtering: This is a filtering process for items that are in the database system. It is used for easy search filtering based on multiple fields. For example, if the user is filtering items based on clock speed and they want to narrow their search further, then they can select another field to filter on. If they decide to filter again on cache size, then they will only see the possible filters for cache size based on the items with the specific clock speed they chose earlier.

2) Gantt Chart:

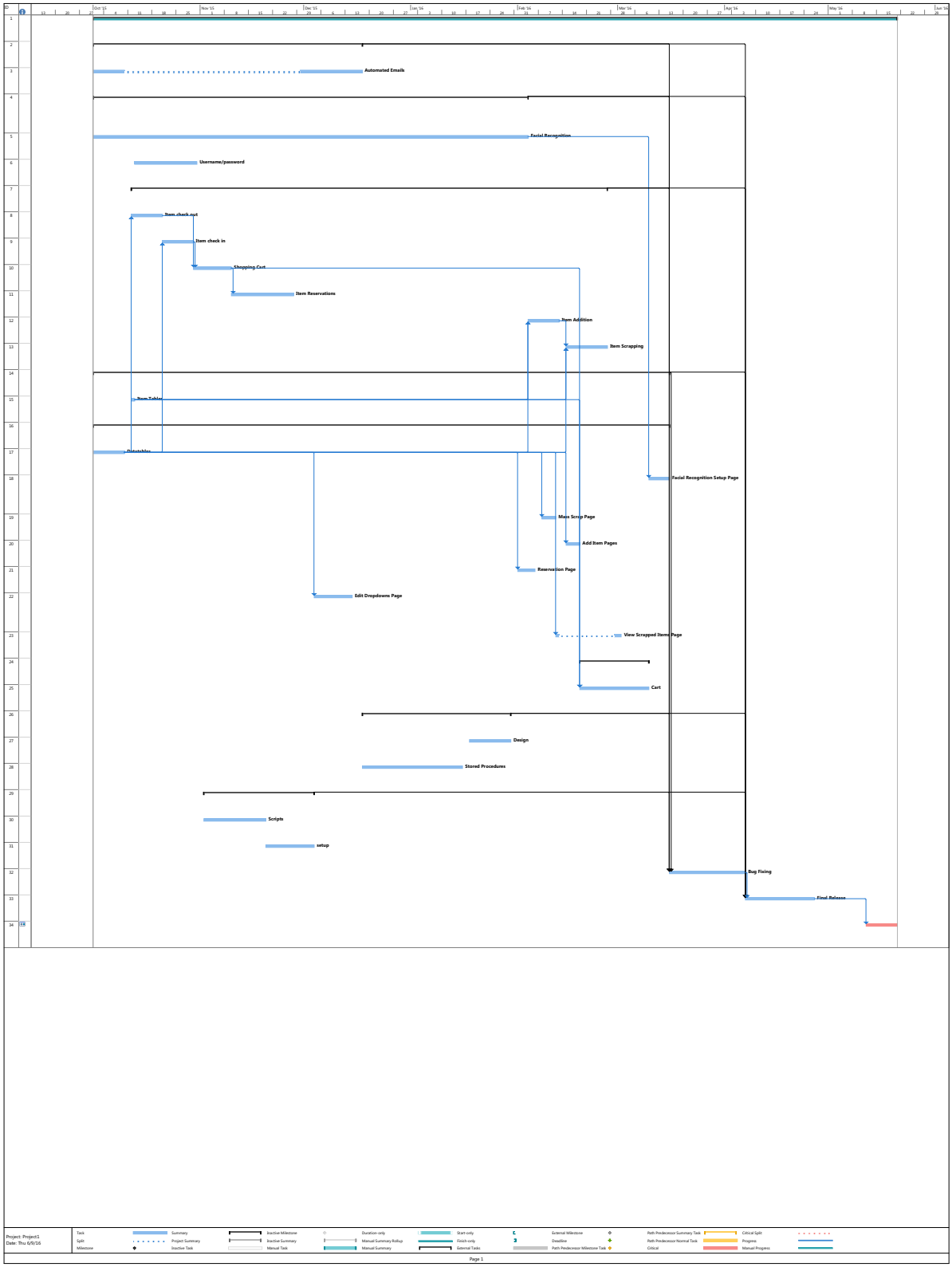

Project: Silicon Tracker
Date: Mon 11/9/15
Reservation system
Item Scanning
Queue Priority
Queue Comments
Item Check-in/out
Inventory Update System
Log-in System
User Authentication System
User Settings
Facial Recognition
RFID
User Permission Levels
Encryption
Item Tracking
Item Retire
Item Insertion
Item Update
Check-out/in Record
Scheduable Reports
E-mail Notifications
Touch Interface
Check in
Check out
Multi Check in
Multi Check out
Add New
Retire Items
Item Search
Display Items
Multi-level Filtering
User View
Fully Functioning Release
Page 1

III. Changes Since the Original Client Requirements Document

#	Requirement	Reason For Changes	Comments
2	Give a reason and a priority when queueing an item. This will create a priority queue for the users. If someone needs an item and it is important, they can give a reason why.	Our client stated that since they are a small team, they only needed an email stating that the item has been checked in. All the people can meet up and talk about who gets the item next.	
10	Retire old items in the system. They need to logically delete (scrap) items when an item is no longer being used. There should be a textbox to optionally give a reason for the item being retired.	There is no textbox to give a reason for the item being scrapped. Each item has a notes field, and we decided that the best option was to place the reason for scrapping in the item's notes field.	
13	Be able to scan an item that is currently checked in to see who has the item reserved next. This will show the next person in line for the item.	We decided not to be concerned about this, because everyone who reserved an item will discuss in person who needs the item next.	
17	If the item is scanned and the item is checked in, it will now be checked out by the user. If the item was already checked out, information about who currently has the item checked out will be displayed on the screen.	Our client asked if we could make it so one person can check in another's item.	
19	When the user is done, they will press a confirm button and will be shown a list of items they have checked in/out. The user can then proceed to commit their transaction to the database.	We made it so they only need to press one button to commit their transaction. A message appears now to just show what items they checked in/out, after they have committed.	
29	Admins will have the ability to enable/disable any of the three authentication methods (username/password, RFID tag, or facial recognition). If for whatever reason the admin does not want one of these methods available, they can disable that method of authentication.	Our client's team implemented RFID security, and the facial recognition was implemented very late in the development phase of the project, so we didn't get a chance to implement this setting.	
N/A	The Interfaces	In our original Requirements Document, we had four interfaces. After starting work on this project, we simplified the system down to two interfaces. It made the system easier to discuss and navigate. We boiled down the interfaces into the Web interface and the Kiosk interface. The Shopping Cart interface, Scrapping interface, and Reporting interface have mostly been combined into the Web interface. The Main Interface has mostly been reworked into the Kiosk interface.	All of the requirements within the Interfaces section have been fulfilled, but not necessarily in all of the interfaces specified.

44 & 45	Will be able to check in items from this interface. If they are at the kiosk they should have a way to check in items using this interface. Will be able to check out items from this interface.	Our client let us know that they only need to check in and check out items from one interface. Our original Requirements Document had checking in and out of items from two interfaces. We removed the Checking in and out requirements to allow for these features to happen in only one place.	
59	Anybody can scan sample RFID cards to check in or out items. We will have sample items for them to scan, and they can use fake RFID badges to simulate the experience.	Our client's team implemented the RFID readers, so we did not show how they worked at expo.	
N/A	New Requirement: Sanitizing and Validation User Input	We did not place in our original document to clean up any user input and make sure that their input is valid. It was an important feature, and our client did ask that we created a system where data entered would be consistent.	
N/A	New Requirement: Editing Dropdown Menu Items	We setup a page for the admins to change the dropdown menu items. These dropdowns are found when inserting or editing an item. Our client liked the idea of not having to make edits directly in the database, so we implemented this feature.	
N/A	New Requirement: Quick Check in/out.	We added an extra feature for when a user scans an item to get the item's information, they can click a Check In/Out button to save the item for later. It makes it easier for the user to check in or out an item as soon as they scan it, and makes the workflow a little smoother.	Our client's team received this feature very well and were thankful to have it.

3) The New Gantt Chart:


Automated Emails
Facial Recognition
Username/password
Item check out
Item check in
Shopping Cart
Item Reservations
Item Addition
Item Scrapping
Item Tables
Datatables
Facial Recognition Setup Page
Mass Scrap Page
Add Item Pages
Reservation Page
Edit Dropdowns Page
View Scrapped Items Page
Cart
Design
Stored Procedures
Scripts
setup
Bug Fixing
Final Release
Project: Project1
Date: Thu 6/9/16
Page 1

IV. The Original Design Document

Below is the original Design Document written at the beginning of the project's lifespan, in its original formatting. It describes our course of action for designing the components, database design, and the workflow of the system. The original document is also included on our USB storage drive.

Design Document

Silicon Tracking Solutions

Client: Scott Oehrlein

Dylan Camus

Brett Hayes

Joseph Cronise

Revisions

Name	Date	Reason for change	Version
Brett, Dylan, Joseph	12/2/2015	Initial Doc	1.0

Table of Contents

Revisions 2

Table of Contents 2

Introduction 3

- Purpose 3
- Scope 3
- Definitions 3
- References 3
- Overview 4

Design Concerns 4

High Level System Architecture 4

Detailed Description of Components 5

- Interface 5
  - Web Application 5
- Server 7
  - Communication between Components 7
  - Scheduled Reporting 9
- Database 10
  - Database Design 10
- Authentication 12
  - Active Directory Authentication 12
  - Facial Recognition 12
- Scanner 13
  - Item Identifier 13

Rationale 14

Introduction

Purpose

This is the design document for the silicon tracking system. This document describes the concerns of the stakeholders and focuses on addressing those concerns by giving a detailed description of how the system will be built. These descriptions are essentially the blueprints of the system. They contain charts, graphs, function headers, and APIs to describe the system.

Scope

The Performance Measurement and Analysis team at Intel in Chandler, AZ desires a new inventory system. Their current system is inadequate for their needs. The Intel team wants a web-based system that can track the items they have as well as scrapped items.

Definitions

Active Directory – A domain controller that organized authorizations and authentications on a domain.

API – Application Program Interface. A series of tools that allow programmers to easily interact with a system.

JavaScript – A high-level language used for the web and other applications.

JSON – JavaScript Object Notation. A standard used for key-value pairs of data.

REST – Representational State Transfer. An interface for communicating over HTTP.

MySQl – A relational database management system.

ER Diagram – A chart that visually represents the relationship between database entities.

References

[1]N. Foundation, 'Node.js', Nodejs.org, 2015. [Online]. Available: http://nodejs.org. [Accessed: 02- Dec- 2015].

[2]A. Reinman, 'Nodemailer', Nodemailer, 2015. [Online]. Available: http://nodemailer.com/. [Accessed: 02- Dec- 2015].

[3]M. Patenaude, 'node-schedule/node-schedule', GitHub, 2015. [Online]. Available: https://github.com/node-schedule/node-schedule. [Accessed: 02- Dec- 2015].

[4] Docs.oracle.com, 'C API', 2015. [Online]. Available: https://docs.oracle.com/cd/A87860_01/doc/network.817/a86082/oidsdk.htm. [Accessed: 03- Dec- 2015].

[5] Software.intel.com, 'Intel® RealSense™ SDK 2015 R5 Documentation', 2015. [Online]. Available: https://software.intel.com/sites/landingpage/realsense/camera-sdk/v1.1/documentation/html/index.html?doc_devguide_introduction.html. [Accessed: 03- Dec- 2015].

Overview

The context of this document begins with a list of concerns from the stakeholders organized by category. The architecture is then described after the concerns. Listed is the high-level overview of the system, followed by a more detailed description of each component. Finally, the justification of our design is given.

Design Concerns

Authentication:

- How will users be authenticated?

Server:

- How will all data be secure?
- How will reporting be handled?

Scanning:

- How will items be identified by the system?

Interface:

- What will the design of the interface look like?
- How will users interact with the interface?

Database:

- What will be the design for the database?
- How will items be retrieved or inserted into the database?

General:

- Will the system be scalable?

High Level System Architecture

Concerns:

- Will the system be scalable?

We have separated this project into multiple components. There is the Interface, Server, Database, Authentication, and Scanner. Having every major section in its own separate component creates a separation of concerns which makes the system scalable. Below are the different major components of the system. Each component will be discussed in more detail in the following section.

- Interface – The front-end component where user interaction with the system occurs.
- Server – The back-end component where the logic of the system occurs.
- Database – Where all persistent data is stored.
- Authentication – Where the user is authorized by different means and credentials checked against the Active Directory server.

- Scanner – Where the items interact with the system.

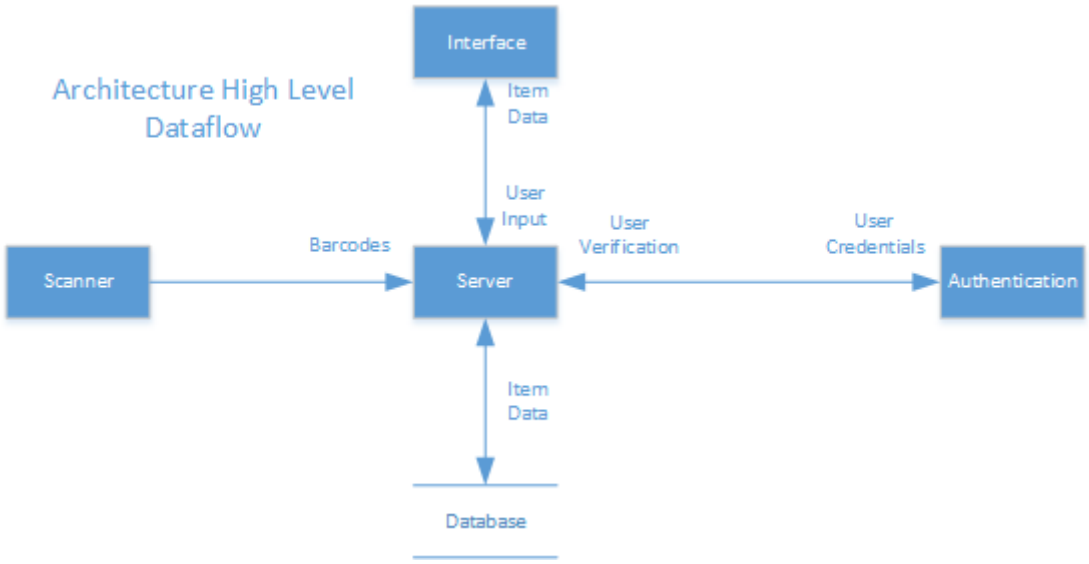


Figure 1: High-Level Dataflow of the System Architecture

Detailed Description of Components

Interface

Web Application

Design Entity Name: Web Interface
Type: Interface
Purpose:
The purpose of the web interface is to allow the user to interact with the database.

Concerns:

- What will the design of the interface look like?
- How will users interact with the interface?

The Web application will be HTML based and the user will be able to interact with it using any standard input method, but it will be designed with a focus on touch screen usage. Users will arrive at the initial login page, where they will select their preferred login method, they will then be redirected to that methods page and will be able to login t the system from there.

Once logged in users will default to the shopping cart interface, and from there they can either use that system, or click one of the tabs to move to a different page. Other available pages will be the user settings page, where users can register their faces for use with the facial recognition login method. The

Reporting page, where users can view, sort, and if they have the correct privileges, make changes to the database. And, the item input page, where users can input new items into the database.

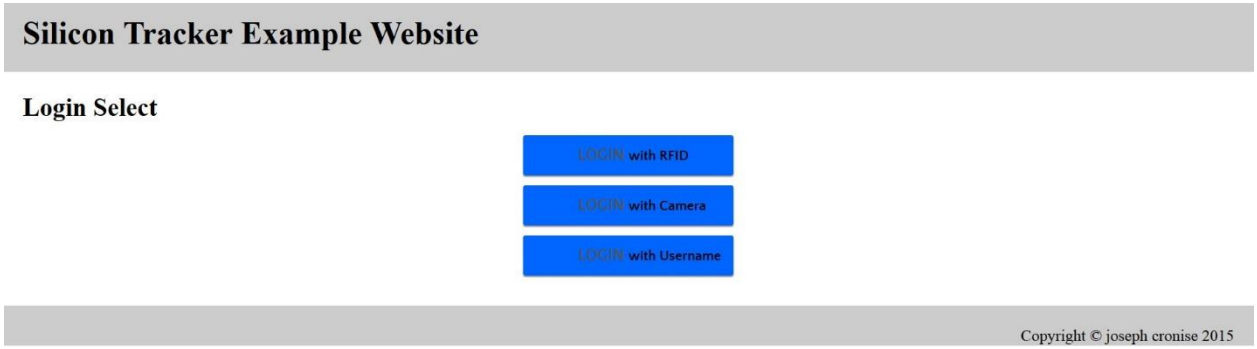
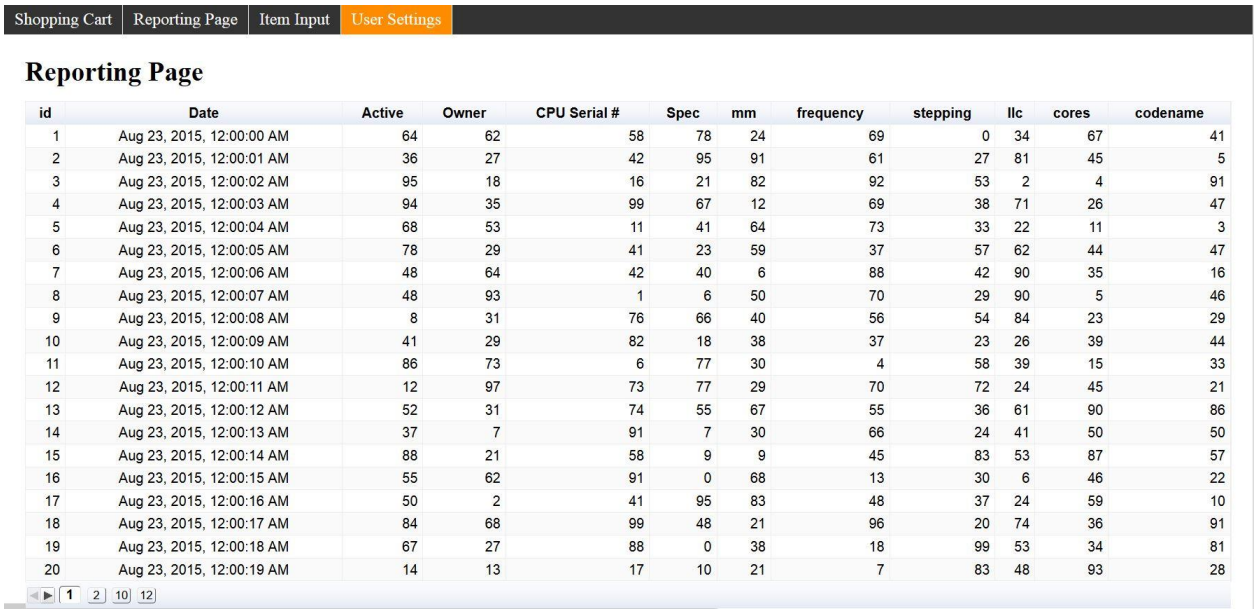


Figure 2: Users will be presented with the option to choose their login method from this page.

Figure 3: If users choose to login using their username/password they will be presented with this style of page.



id	Date	Active	Owner	CPU Serial #	Spec	mm	frequency	stepping	llc	cores	codename
1	Aug 23, 2015, 12:00:00 AM	64	62	58	78	24	69	0	34	67	41
2	Aug 23, 2015, 12:00:01 AM	36	27	42	95	91	61	27	81	45	5
3	Aug 23, 2015, 12:00:02 AM	95	18	16	21	82	92	53	2	4	91
4	Aug 23, 2015, 12:00:03 AM	94	35	99	67	12	69	38	71	26	47
5	Aug 23, 2015, 12:00:04 AM	68	53	11	41	64	73	33	22	11	3
6	Aug 23, 2015, 12:00:05 AM	78	29	41	23	59	37	57	62	44	47
7	Aug 23, 2015, 12:00:06 AM	48	64	42	40	6	88	42	90	35	16
8	Aug 23, 2015, 12:00:07 AM	48	93	1	6	50	70	29	90	5	46
9	Aug 23, 2015, 12:00:08 AM	8	31	76	66	40	56	54	84	23	29
10	Aug 23, 2015, 12:00:09 AM	41	29	82	18	38	37	23	26	39	44
11	Aug 23, 2015, 12:00:10 AM	86	73	6	77	30	4	58	39	15	33
12	Aug 23, 2015, 12:00:11 AM	12	97	73	77	29	70	72	24	45	21
13	Aug 23, 2015, 12:00:12 AM	52	31	74	55	67	55	36	61	90	86
14	Aug 23, 2015, 12:00:13 AM	37	7	91	7	30	66	24	41	50	50
15	Aug 23, 2015, 12:00:14 AM	88	21	58	9	9	45	83	53	87	57
16	Aug 23, 2015, 12:00:15 AM	55	62	91	0	68	13	30	6	46	22
17	Aug 23, 2015, 12:00:16 AM	50	2	41	95	83	48	37	24	59	10
18	Aug 23, 2015, 12:00:17 AM	84	68	99	48	21	96	20	74	36	91
19	Aug 23, 2015, 12:00:18 AM	67	27	88	0	38	18	99	53	34	81
20	Aug 23, 2015, 12:00:19 AM	14	13	17	10	21	7	83	48	93	28

Figure 4: A sample Reporting page showing a possible output from the CPU table.

Server

Communication between Components

Concerns:

- How will items be retrieved or inserted into the database?
- How will users be authenticated?

The server is the central component to the system. All communication between components and outside systems will go through the server. In order to keep the system as loosely coupled as possible, there should be a minimal amount of entry points between components.

The high level architecture figure shows the data traveling between components. This section will go into greater detail on how these components will communicate through the server.

We have decided to use Node.js as our server system. Node.js is an event-driven system and is efficient at asynchronous calls. Node.js is also great at including packages created by other developers, such as a mailing agent, scheduler, etc.

Design Entity Name: Interface-Server Communication
Type: Events
Purpose:
The interface is how the user communicates with the system. This is the entry point for any user. This is also the entry point for human error input and possibly even malicious attacks on the system. The basic idea for connecting the interface with the server looks like the following:

- The user sends input data
- The server sends information from the database, and error messages if need be.

We will be implementing a RESTful API for the client and server to interact with each other. The pages will be as follows:

/ - The home page.

/shop - The shopping cart.

/scrap - The scrapping cart.

/reporting - The reporting page.

/settings - The user settings.

Many of the requests between the interface and server will be GET requests. This will be in the form of item information including data of the user who has checked out the item, or user settings.

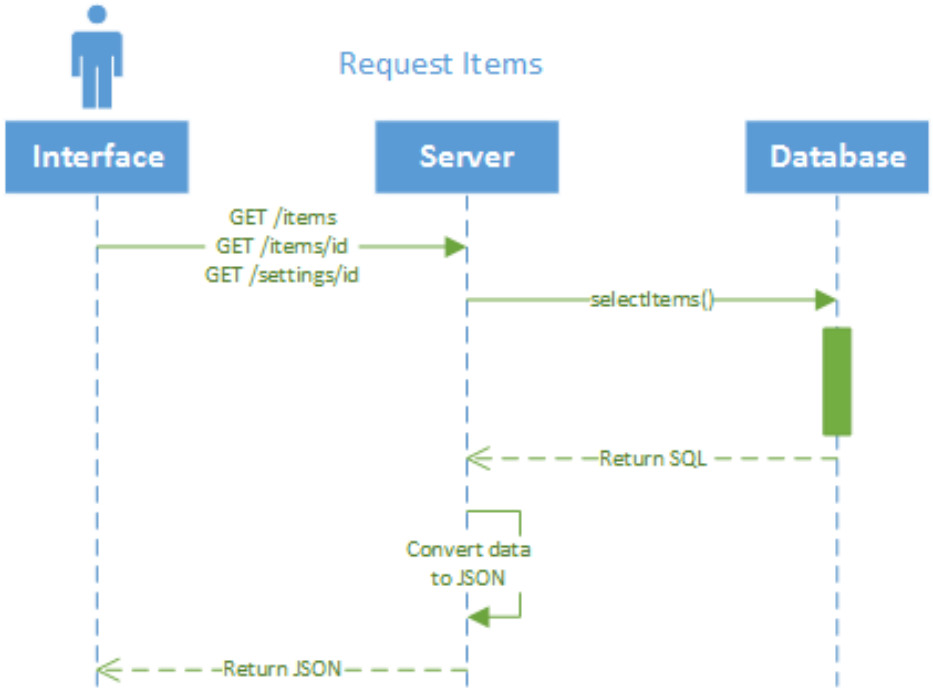


Figure 5: REST Sequence Diagram

Design Entity Name: Database-Server Communication
Type: Events
Purpose:
There will be two main types of requests from the server to the database. The first is the set of requests coming from the interface. The second are scheduled reports handled by the server.

There will be SQL scripts stored on the server. These scripts will be dynamic so that depending on user input the correct information will be returned from the database. For example, a user requests all items that have an L1 cache greater than 32K. The script would include this in the WHERE clause to filter the results returned.

These scripts will be called by a function and that function is in charge of interacting with the database. The function will send the script to the database and after the database sends the results back to the server, the function will convert the data to JSON and return the results.

Function Name: ExecuteQuery
Parameters:

- Script - The script to be executed

Returns: JSON – The results from the database in JSON format

Design Entity Name: Authentication-Server Communication
Type: Events
Purpose:

Whenever a user needs to use the system, they will have to go through authentication in order to login to the system. We will be implementing an API that allows the server to interact with the Active Directory server.
[Add reference to Dylan's work here]

Scheduled Reporting

Concerns:

- How will reporting be handled?
- How will all data be secure?

The server will be in charge of sending out scheduled reports to users as well as unscheduled reports based on triggers. Examples of scheduled reports would be: reminders of items checked out. Examples of unscheduled reports would be: receipts after checking in or out items; mass e-mail sent by admin.

Design Entity Name: E-mail Agent
Type: Resource
Purpose:
The server will need an effective way of sending e-mails to the users. This will provide a quick and efficient way of compiling and sending e-mails to the correct users.

We decided to use Nodemailer, a mailing agent that works with the node.js server. With Nodemailer, we will be able to send out e-mails securely through SSL. We will also be able to easily create custom e-mails with the right information for the users.

The mailing agent can be set up with the following options for the mailer connection data:

- options.port is the port to connect to (defaults to 25 or 465)
- options.host is the hostname or IP address to connect to (defaults to 'localhost')
- options.secure defines if the connection should use SSL (if true) or not (if false)
- options.auth defines authentication data (see authentication section below)
- options.name optional hostname of the client, used for identifying to the server
- options.localAddress is the local interface to bind to for network connections
- options.connectionTimeout how many milliseconds to wait for the connection to establish
- options.greetingTimeout how many milliseconds to wait for the greeting after connection is established
- options.socketTimeout how many milliseconds of inactivity to allow

There are more options available and are displayed in the Nodemailer documentation.

Design Entity Name: Scheduler
Type: Resource
Purpose:
The scheduler is used for sending out the scheduled e-mails to their respective users.

Like the mailing agent, we are using a package developed by an outside programmer in order to implement the scheduling agent. For scheduling, we will be using a package called node-schedule. This

system makes it simple to schedule tasks. The main method we will use is a date and time scheduler. The method takes in a JavaScript Date object, and executes a task once that date has been reached. A scheduled event can also be cancelled at any time.

Database

Database Design

Concerns:

- What will be the design for the database?

Design Entity Name: Inventory Database
Type: Resource
Purpose:
The Inventory Database is used to track items within the lab.

We will using a MySQL database to track different items within the lab. The database will also keep a log of all item check outs/ins and will be used as part of the item reservation system where a user can reserve an already checked out item. The database will work with the server in order to send out reports as well as allowing manual checks of the data.

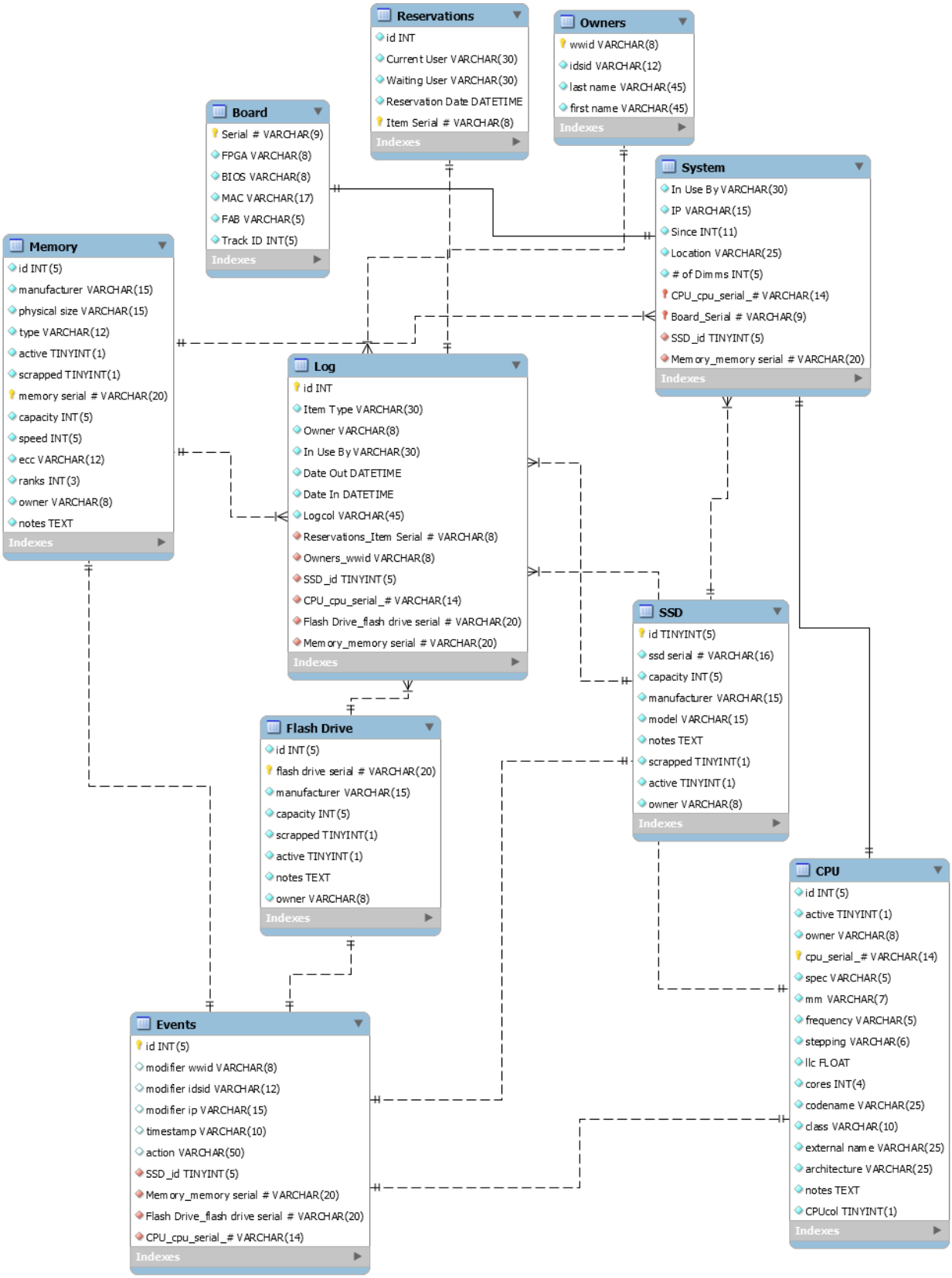


Figure 6: ER diagram detailing the specifics of the Inventory Database

Authentication

Active Directory Authentication

Concerns:

- How will users be authenticated?

One of the ways users will be authenticated is by entering in their security credentials or sliding their RFID and comparing with Intel's Active Directory. The Active Directory API allows an easy method of sending these credentials to the Active Directory securely with encryption.

Design Entity Name: Active Directory API
Type: API
Purpose:
The Active Directory API specifies how the system architecture interacts with the Active Directory Server. The Active Directory server contains security credential information that needs to be referenced with the information gathered from the system. The responsibilities of the Active Directory API are to take a number of parameters and return back a value that indicates either success or failure of authentication. The data sent to active directory must be encrypted with SSL. The machine on which the system is run on must contain a properly formatted certification authority certificate that matches the certification authority of the active directory.

The subcomponents of the Active Directory API are the RFID_authenticate function, which takes an RFID data field as a parameter and returns either success or failure; and a credential_authenticate function, which takes a username and password and returns either success or failure. The Active Directory API interacts with the system server and the Active Directory server. The system server listens for user input from the interface. Once input is received, the system server encrypts the data and sends it to the active directory server using the Authentication API. The Authentication API then returns either a success or failure to the server.

The Active Directory API requires both a keyboard for inputting security credentials and an RFID scanner. It also requires access to the Active Directory Server. The authentication API performs its task by first receiving both username and password in the case of credential_authenticate, or the RFID data in the case of RFID_authenticate. First, ldap_initialize is called to initialize a connection to the Active Directory Server. Then, ldap_simple_bind_s is called with username and password or the RFID data.

Facial Recognition

Concerns:

- How will users be authenticated?

Facial recognition will be a method of authentication separate from the active directory. After logging in, a user may choose to set up facial recognition. This allows users to quickly log in without needing to type in their user credentials or swipe their RFID badge.

Design Entity Name: Facial Recognition API

Type: API
Purpose:
The Facial Recognition API specifies how the system architecture interacts with the Intel RealSense Camera. The Intel RealSense Camera possesses facial recognition capabilities that, once properly initialized, will allow users to quickly authenticate themselves without having to use the keyboard to type in their security credentials. The responsibilities of the Facial Recognition API is to register users for facial recognition in the database, and to perform facial recognition using the images stored in the database. The API must have access to the Intel RealSense Camera API. Additionally, the API requires access to the database for storing images of registered users. Finally, users must have logged in using either their security credentials or RFID before being able to register as a new user for facial recognition.

The subcomponents of the Facial Recognition API are the recognition_init function, which initializes the RealSense Camera for Facial Recognition and specifies where to store the image files created when registering new users; the recognition_register function, which registers a new user for facial recognition by taking a picture of the user's face and storing it in the database; and the recognition_authenticate function, which compares the users face with facial images stored in the database.

The Facial Recognition API interacts with the system server and the database. The API is called by the server when a user wishes to register as a new user or authenticate him or herself using facial recognition. The API then interacts with the database by either writing facial images or reading facial images from the database.

The only outside resources required by the Facial Recognition API is the Intel RealSense Camera itself and the database. The database must have generous storage capabilities to store multiple JPEG files per user. The Facial Recognition API performs its task by making use of the Intel RealSense API for facial recognition. When initializing facial recognition using recognition_init, the RecognitionConfiguration interface from the Intel RealSense API is used to enable face recognition, specify the database used for storing the facial images, and setting the registeration mode. For registering a new user, the recognition_register function makes use of the RegisterUser function from the Intel RealSense API, which takes a picture of the user and stores the image in the database. Finally, the recognition_authenticate function makes use of the RecognitionData interface of the Intel RealSense API to parse all images stored in the database and calls the QueryUserID function to compare the users face with the images stored in the database.

Scanner

Item Identifier

Concerns:

- How will items be identified by the system

A major concern of the system is how to identify items in an efficient manner. Users will potentially need to identify many items in a single transaction, and therefore a scanner has been chosen as the resource for this purpose as it allows the system to avoid having to go through the interface for each item in a transaction.

Design Entity Name: QR Scanner

Type: Resource
Purpose:
The QR Scanner is used to check items in and out of the database. The scanner reads a small QR code printed on the items. The scanner is useful in that it allows users to quickly identify an item, such as a piece of silicon, from within the database, list information about the item to the user, and automatically update its checkout status. The scanner works specifically with the server, which holds a list of all QR codes scanned by the user during a particular session. Once the user is finished scanning and ready for checkout, the server then writes the new checkout status to the database for each item scanned.

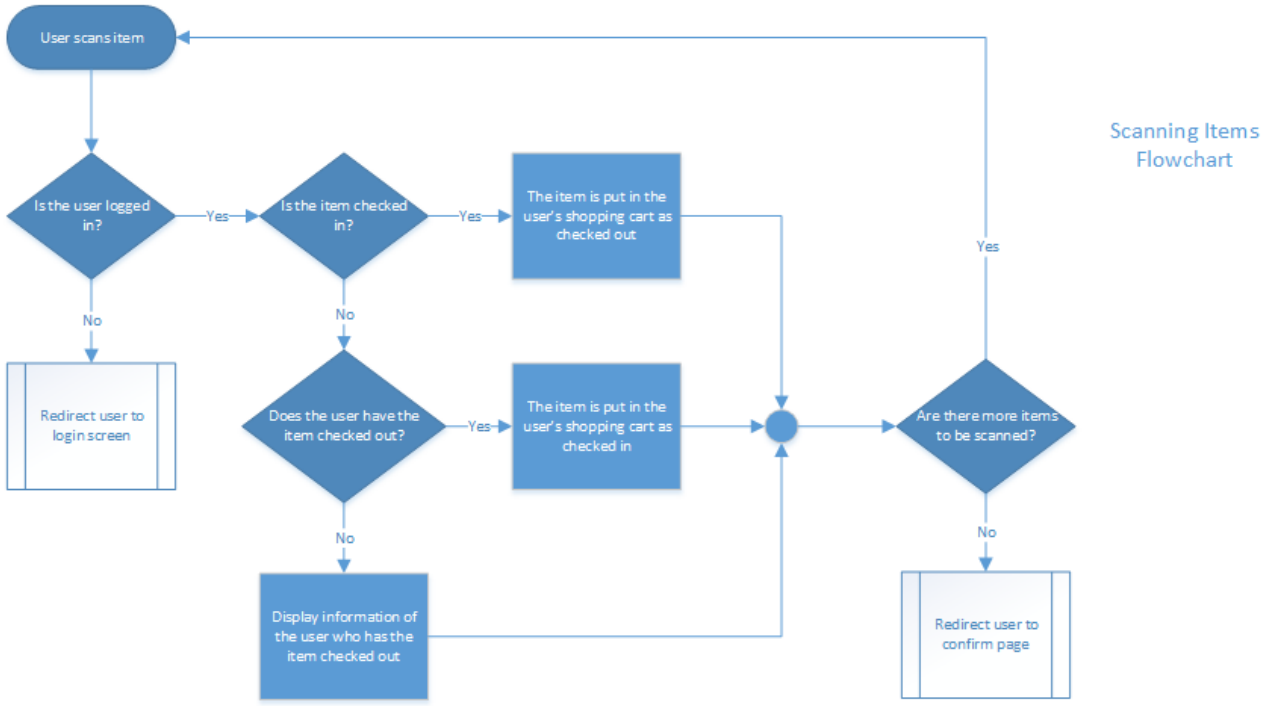


Figure 7: Flowchart detailing the decision process when scanning an item

Rationale

Our client had a clear vision of what he wanted in this new inventory system. He knew he wanted a web-based system that could scan items, work with their Active Directory server among other specific details. These details guided our decision-making when designing the system.

V. Changes Since the Original Design Document

We have changed the design of the high-level architecture. When we first started designing the project, we weren't sure on how the scanner would work. We had thought that we would have to design a separate component for the scanner to handle events and sent barcode information to the server. After playing with the scanner, we learned of its behavior and was able to handle the barcode scan as an event inside the Interface component. The scanner simulates a keyboard and presses a key for each character of the barcode it just scanned. We were able to systematically determine when a barcode was scanned and when a human was typing on the keyboard, thus having no need to separate the scanner into its own component.

Along with the scanner being a part of the Interface component, we also moved the Authentication component into the Interface component. We basically created only a single component that the user can have direct access to. This created a much simpler design and made it easier to develop, since there were less components all needing to communicate with the server.

We had originally thought we would need to create a system that authenticates users based off an Active Directory server. After creating this document, we found out that the system would need to just send a web request to authenticate. We weren't told any more information, since that would be a security breach. We ended up creating a mock authentication system and we let our client's team handle the security from there.

In the Interface component, we had split the design work into two subcomponents. These subcomponents are the the Web interface and the Kiosk interface. The Web interface is for monitoring and viewing large sets of information, adding, editing, or removing items, etc. It is designed to be used for any work that would be easier to use a computer with keyboard and mouse. The Kiosk interface is designed in a way to be touch friendly, and to handle quick transactions, such as the checking in and out of the silicon chips.

The database schema has changed since the original design. We originally had a table for each type of item. Today we still have those item type tables, but we have a more generic table that specifies what the item type is, and references data in the more specific tables. The generic table, Items, holds the barcode, the notes for the item, if the item is checked in, and if the item has been scrapped, or logically deleted.

After a large amount of research and help from mentors, we had discovered that the SDK for the Intel RealSense Camera was not compatible with facial recognition on a web browser. This meant that we couldn't use the facial recognition software that was designed for this camera. We had to instead take two images using the RealSense camera. The camera takes an RGB image and a depth image, and uses Open Source Biometric Recognition (OpenBR), a runtime library for image processing, to do facial recognition on the RGB photo and an image comparison for the depth image.

VI. The Original Technology Document

This is the Technology Document we wrote at the beginning of the project. It has been reformatted to match the styling of this document, but the content of the document has not been modified. The original document in its original formatting is included on the usb storage drive.

A. What We Are Trying to Accomplish

The PMA team at Intel is in need of a new inventory tracking system for their silicon and other items they need to store. They would like a web-based inventory system where engineers can check in and check out items. All the items tracked will need to be stored on a database. This database will need to track items and user settings. The application will also need to communicate with the Active Directory server at Intel for user verification. Every inventory item has a QR code that needs to be scanned when it is checked in or out. When an item is scanned, the inventory system will update the database accordingly.

B. Cameras

1) Standard CCTV camera: The first possible option for the facial recognition camera would be to use a standard CCTV camera, this has the benefit of offering a wide range of products at varying price points, as well as allowing usage of a large number of pre-existing facial recognition software's. However, many of those software's are costly, and are more limited than an algorithm based around the capabilities of the next two options. A CCTV camera on its own has limited functionality.

a) Cost: Varies

2) Intel RealSense Camera(F200): The RealSense camera is Intel's 3d camera designed for laptop and tablet integration. This Camera was designed to allow for gesture control, facial analysis, sound processing, and augmented reality. In terms of facial recognition, it has a range of 25-75cm. The camera has three display modes, Depth, IR, and RGB. The standalone developer version requires USB 3.0, but the final product would make use of an integrated camera. It supports windows 8.1 and newer and requires a 4th generation (or later) Intel Core processor. This camera has depth precision fine enough to detect gender, emotion, and even the users pulse. The main drawback is the products relative newness, meaning its developer community is much smaller as compared to the other options.

a) Cost: standalone dev kit: $99.00

3) Kinect Sensor: The Kinect Sensor is another possible option for the facial recognition camera. This sensor, originally designed for Microsoft's Xbox 360 and significantly improved upon with the release of the Xbox One provides much the same functionality as the RealSense camera. It is a depth sensing camera capable of multiple modes, Color and IR, with a stated range of 5 meters. It was designed with full motion body tracking in mind, though it is fully capable of performing facial recognition. At 50cm the quality would be .75 mm per pixel. Historically the Kinect Sensor has had issues recognizing individuals with dark skin, though Microsoft has blamed those results on lighting levels. The sensor is also designed with the Xbox as its primary usage, there was a dedicated windows version, but that was discontinued in favor of a simple adapter for the Xbox version. The Camera is USB powered and has a robust developer community.

a) Cost: $99.99 + $49.99 adapter

4) Selection Criteria: The chosen camera must be able to output an image with enough detail to successfully recognize an individual. The output must be clear enough to differentiate between an image of a person and the actual person, and the Camera needs to be able to integrate with current systems already in place. Due to project criteria, and the specific request of the sponsor, the camera selected was the Intel RealSense Camera.

C. RFID Reader

1) 1126 Desktop UHF RFID Reader: This product is USB powered and offers both read and write capability for RFIC cards. It is specifically designed for PoS, document tracking, and access control applications and has an extensive SDK. It has the benefit of providing both audio and visual confirmation of a successful read and its built in antenna has a range of up to 1.5 meters, tag/environment dependent. It is designed as a desktop device and as such has no built in mounting bracket or the ability to add one. It works with Windows, Mac, and Linux and is a sturdy, drop resistant design.

2) ID CPR02.10-AD/-B RFID Card Reader: This is a wall mounted reader which requires an external power supply. It is unable to write cards and has two possible communications links. Serial (RS232 or RS485) or Data/Clock (Wiegand). It has a max read distance of 7cm and includes both visual and audio confirmation of a successful or unsuccessful read. It includes both the reader, and the wall- mounted housing. It functions with Windows, Mac, and Linux.

3) pcProx82 Series: This is a USB powered device which works with nearly all proximity, contactless, and magnetic stripe card technologies. It is designed to add additional functionality to already existing employee ID badges by integrating them into applications beyond simple door access. The main benefits of this technology are its versatility, ease of use, and the fact that it has no license restrictions. It is compatable with recent windows operating systems and is capable of sending data as serial ASCII or non-keystroke formats with an average maximum read range of 2.5-7.6cm. It includes a Tri-state LED and beeper for read confirmation and comes with a 1 year material/workmanship defect warranty.

4) Selection Criteria: The reader must be able to successfully read all relevant data from a standard Intel ID badge as well as integrate properly with a tablet PC. Due to project criteria, and the specific request of the sponsor, the RFID reader selected was the pcProx82 Series RFID reader.

D. Database Management System

1) MySQL: This is an open source relational Database Management System (DBM). It is one of the most widely known client-server DBM. One of the main benefits of MySQL is that it is already integrated into existing open source softwares such as WordPress and Drupal due to it being a part of the Linux Apache MySQL PHP (LAMP) package. Additionally, MySQL is supported on nearly every operating system and supports many programming languages. MySQL is known to be easy to setup and use and is the industry standard DBM. However, MySQL is known to have stability issues and suffers from poor performance scaling.

2) Microsoft SQL Server Express: This product is the free version of Microsoft's SQL Server. The express version is open to distribute and use, similar to open source software. It implements many of the same features as MySQL but with a better query optimizer, the T-SQL procedural language, and a window-based management tool. However, due to being the express version, the DBM has some limitations. For example, there is a maximum size of 10 GB per database. Additionally, there are artificial hardware usage limits such as only one physical CPU and 1 GB of RAM can be used. Finally, SQL Server Express is limited to running only on Windows and supports only .Net, Java, PHP, Python, Ruby, and Visual Basic.

3) Oracle Database Express: Oracle is an object-relational database management system, meaning that it supports objects, classes, and inheritance directly in the query language. It is an express version and, like SQL Server Express, it has limitations set on data, allowing only 4 GB. Oracle Database is generally targeted at large enterprise applications for huge corporations. It is supported more operating systems and supports more languages than either MySQL or SQL Server.

4) Selection Criteria: Due to the nature of the project, licensing is very important. Therefore, MySQL is the clear answer due to it being open source. Additionally, since MySQL is inherently free, it does not have any artificial limitations imposed on it like the other options have. Microsoft SQL server and Oracle Database both lend themselves to large enterprise applications, whereas MySQL is generally used for smaller web applications. Finally, the team has much more experience with MySQL than the other two languages.

E. Web Application

1) Node.js / Angular.js: Node.js is a javascript runtime engine that runs on Google's V8 engine. We could use this as our web server. Node.js would handle any server-side functionality, and Angular.js would handle the client-side interface. Some of the advantages are that the system would be very scalable, testable, and reliable. Both Node and Angular are built for event-driven, non-blocking systems. This is particularly useful since we will have a number of I/O devices working with our application. It would also be able to run securely with SSL. There are a few downsides to these technologies as well. There is a learning curve that would be particularly larger since we would be learning two technologies at the same time.

2) Visual Studio Community: This is a free version of Visual Studio that we could use to create the web application. VS Community is a full IDE with everything we would need to create a fully interactive web environment. Building an ASP.NET application would be a good option because it is such a powerful framework. The back-end would be written in C#, which is designed in a way to easily interact with databases, lists, string manipulation, and much more. It comes with Linq, which allows developers to create database queries, and sort through lists seamlessly. This is also a very large framework, and the learning curve is probably greater than any other option listed. There are also limitations to the licensing with the free version, so that also has to be kept in consideration.

3) Vanilla HTML/CSS/Javascript with PHP: A third option would be to go without a web framework, and use a more basic approach to developing the software. We would just write base web interfaces with HTML, CSS, and Javascript, and then use the server-side tool PHP, which is a more minimal scripting language. This would benefit us by giving us the freedom to come up with whatever design structure we want to use. We could essentially write the code in anyway that we feel would be best for the system without being imposed by framework rules and guidelines. On the other side of that argument is that we could have too much freedom and create a complex and chaotic system that is nearly impossible to understand, since we didn't impose any coding standards. It would have a smaller learning curve at the beginning, but as the system grows it might not be the best option.

4) Selection Criteria: As stated in other sections, licensing is very important. All three of these systems are free, but Visual Studio Community will have some limitations that would deter us from using it. We also have to make sure that we can read information from I/O devices, such as the RFID reader and the camera. Node.js has the ability to work with I/O systems easily. It has the ability to run C and C++ libraries within the system, which is useful for the RFID readers and cameras. Angular.js easily integrates with Node.js and can create dynamic web pages.

VII. Changes Since the Original Technology Document

Because of a large learning curve for software that wasn't necessary for development, we decided not to use Angular.js, a javascript web framework, for the interface. Instead, we used only HTML, CSS, and JavaScript/JQuery for our interface, without a framework.

We had discovered that the SDK for the Intel RealSense Camera could not do facial recognition if the camera is being used through a web browser. Because of this, we switched to using Open Source Biometric Recognition (OpenBR), a runtime

library for image processing.

VIII. Weekly Blog Posts

Throughout the course of the project, we wrote a weekly update on a blog post whenever there was work done on the project. Each blog post is in chronological order and styled in a way that makes it easy to differentiate between them.

A. Welcome to my blog! by Cronise, Joseph Eric

This is where I'll be sharing my thoughts on topics that matter to me. Who knows... I might even share pictures, videos and links to other interesting stuff.

If I catch your interest, let me hear from you.

B. Weekly Update 10/23/2015 by Cronise, Joseph Eric

This week we held our weekly meeting with Scott Oehrlein on monday. We went over his changes to the design document as well as clarified a few matters and asked any questions we needed.

On tuesday we were given two Realsense cameras which were distributed to Dylan and Brett, Joseph has his own, albiet an older model. Each member took some time to briefly familiarize themselves with the camera.

Over the entire week we, individually, have been looking into different web framework options for the project as well as refamiliarizing ourselves in database operations.

C. Weekly Update 10/30/2015 by Camus, Dylan Michael

Again this week we held our weekly meeting with Scott Oehrlein on monday. We didn't have too much to talk about but we discussed the requirements document and planned to meet with Scott in person on Wednesday November 4th.

We have continued looking at web frameworks and database options and are starting to get a much better handle on what the project will look like.

D. Weekly Update 11/6/2015 by Camus, Dylan Michael

This week our weekly conference call with Scott was canceled because our client Scott Oehrlein was in Corvallis and we were able to meet with him in person. We went over the old web app that the Intel team currently uses and where he wanted to see improvements. We also received three RFID scanners from Scott to work with.

In addition, we had our first meeting with the TA this week. We went over our requirements document and where it needed improvement.

E. Weekly Update 11/13/2015 by Camus, Dylan Michael

We had a little trouble this week due to daylight savings time. We ended up missing our weekly meeting with Scott because Arizona does not recognize daylight savings time. Additionally, because of veterans day school was canceled, and therefore we did not meet with our TA either.

What we did, however, was complete our revisions on the requirements document and finished the tech review. We are looking to begin working on our design document soon.

F. Weekly Update 11/27/2015 by Hayes, Brett

This week has consisted of a number of different tasks.

We talked with Scott about a week ago about adding/deleting a few user stories from the requirements document. We have bounced ideas back and forth with our client and his needs for the system. We have a new revision of the document and hopefully we this version has everyone happy.

We still haven't had success with the RFID cards and readers. The cards from our client are blank, and we are finding it difficult to find anyone with an RFID writer so we can put fake data on the cards. We tried the RFID readers with our student IDs but have not had any success with them.

We prepared our elevator speech and are ready for our turn next week to give the speech. Brett is giving the intro, Dylan is giving the problem and Joseph will deliver the solution.

We plan on working on the design document this weekend and will work hard on it in the beginning of next week.

G. Weekly Update 12/5/2015 by Hayes, Brett

We had a lists of concerns for this project. We placed our list of concerns into the design document and wrote about how we plan to address those concerns.

Brett mapped out the high-level design and how data is going to flow throughout the system in a brief but informative way. He also went into detail about how the server will be constructed and some design features we will be implementing.

Joseph came up with some prototype designs for the interface. He was able to create a model of the system and what it will look like. Some screenshots of the mock-up are in the design document. Joseph also created a mapping diagram of the database.

Dylan wrote an interface for the Active Directory API and how we will be working with the RFID reader, facial recognition, and username/password authentication. He also wrote how the barcode scanner will be used and implemented into our system.

We spent most of the week on the design document. Aside from the document, we received the barcode scanner from Scott and were able to test out the scanner on some fake CPU silicon chips that Scott also provided.

H. Weekly Update 1/16 by Hayes, Brett

The previous week was geared towards setting up our coding environments. This week we have been building the basic blocks of our product.

Dylan continues to work on the database scripts. He is writing what will be the CREATE TABLE scripts.

Joseph has been developing a skeleton of all the web pages for the project. The welcome screen has been placed on our webpage. It contains a simple username and password form.

Brett has set up the database that the group will use to develop the project. It is connected with the hosted website that he put up online. He is also looking into methods of how to detect when a barcode scanner and an RFID reader will be hooked up to the kiosk.

We had our weekly meeting with Scott on the 15th. We decided that next week we will be going over the design of the interface considering we get a few more pages online. Scott plans on visiting OSU in February, so we also discussed a good meeting time for when he is in town.

I. Weekly Update 1/29 by Hayes, Brett

Joseph has been working hard on the interface, and making it look aesthetically pleasing. He has been the primary person to work on the front-end coding. Joseph has also been working on figuring out what is needed for the RFID tags.

Dylan has continued to work on the database scripts. After writing the scripts to create the tables for the database, he has been working on the stored procedures that will be called when creating an item, looking up an item, etc.

Brett has been working on the server-side code. He has been attempting to make the website secure through SSL security. Brett has also started looking into what it will take for an automated e-mailing system. He has begun the process of integrating OAuth authentication.

After our weekly meeting with Scott, we have decided to rewrite the styling of the website. He was also able to send us some sample data after the meeting so that we can begin testing some of the scripts we have written.

J. Weekly Blog Post 2/5 by Hayes, Brett

Joseph has continued to work on the interface design. He has made a lot of progress and is now waiting for our client Scott to review.

Dylan and Brett have been working together on the server-side code. They have made the server connect to the database and have the ability to pull data for display on the website. Brett has also made a table with multi-column filtering (one of the requirements).

The next steps are building a check in/out system, writing the midterm report, and recording a video for the midterm. Our hope is to get a somewhat polished version of our website before Scott arrives to Oregon on 2/17.

K. Weekly Update 2/12 by Camus, Dylan Michael

Our client Scott reviewed the updated interface design that Joseph created and was overall satisfied with the design. However, he has some concerns over the usability on smaller resolutions.

Dylan and Brett continue to update the server code. They are working on getting the SSL working and adding functionality for adding items to the database through the application. Additionally, work is being done on the kiosk app to allow for checking items in and out.

We finished our midterm report and video. Moving forward, we are looking to have the core functionality of adding items and checking items in and out ready for when scott comes on the 17th.

L. Weekly Blog Post 2/19 by Hayes, Brett

Scott came to visit from Arizona to see the progress we had made on the site. We actually came up with a lot of progress in the past week.

We now have the kiosk inserted into our live site. If a user wants to view the kiosk, they will have to type in the navbar after the website '/kiosk'.

A user is now able to scan an item when in the kiosk interface. If the user is on the login screen, they can scan an item and the information about the item is returned for viewing. The same code will be used for implementing the check in/out sytstem.

In our meeting with Scott and his team (through Skype) we were able to show the current progress of our system. Scott and his team had some great feedback, and it was really good to hear what they had to say. We have a lot of details about their wants and needs in the system.

M. Weekly Blog Post 2/26 by Hayes, Brett

Brett worked on server administration mostly. He moved the repository to a public repo on github. He also spent time writing the readme on the github page explaining how to deploy the app onto a local machine, or with little modification, to deploy to a server. The new repo can be found here:

https://github.com/bretterism/silicontracker

Along with writing the directions and moving the repository, Brett was able to move the latest production environment to a home server. The reason for this was that our hosting space on OpenShift was handling a lot of the SSL security automatically, so we want to make sure that our app works specifically with SSL and implementing whatever we need for that.

We also didn't have our weekly meeting with Scott. He had to cancel, and we all had previously decided to set another time to get together anyways. We are currently figuring out what time works best for everone.

N. Weekly Blog Post 3/4 by Camus, Dylan Michael

This week we worked on getting the last of the features that need to be in before the beta implemented. Brett set up the website on a home server and set up ssl security. He also set up the github repository for our migrated code and is working on updating the checkin/out system with user profiles.

Dylan updated the database to have a more efficient way to store which items are curretnly checked out. He also setup an email scheduler that sends emails over an interval. He is currently working on creating a SQL stored proceedure to only send emails to users who have had an item checked out for some length of time.

Joseph has been working on the facial recognition authentication. He currently has it so that the web app will select the users web cam and display the feed on the login page. He has looked into implementing the facial recognition algorithm and has it almost finished pending more testing.

Our meeting times with Scott have been rescheduled to 9:30 AM on Mondays.

O. Weekly Blog Post 3/11 by Hayes, Brett

This week we discussed a few of the features with Scott, and he responded with good feedback. We added a feature to add new attributes for the dropdown lists when adding a new item into the system. A dynamic table is loaded to create edit or delete these dropdown attributes whenever there is a change to be made.

We also added a user settings page, which lets the users change some basic features if they desire. This is where the dropdown attributes are found.

Basically we have added a way for the user to login, and interact with the system as a user. We implemented sessions for user. Basically, the user's information is kept on the server and is passed to the client on a per-need basis. That way secure information, such as the user's WWID, can never reach the client, but their name and user settings can be reached.

This next week we plan to go through our requirements document and make sure we have all the requirements fulfulled. We will be working hard on writing the document and will get together in the next few days in order to create our video for the final presentation.

P. Weekly Blog Post 4/8 by Hayes, Brett

Starting back into the term, we have worked very hard at refactoring a lot of our code. At the end of the winter term, we had pushed a lot of code, so our code got pretty messy. We refactored it over spring break and started the term off with a clean code-base.

Part of the refactoring involved stripping the web template we were using for our web pages. The template did not look well on all screen sizes, so we stripped the template and have a responsive web site.

We have derived a task list that covers all the code-related tasks for this term. Our task list can be found here:

https://docs.google.com/spreadsheets/d/1ue5x-tGK24JxzuKh7rXZGm6fWu7G3c6rAJPGCW_5Hcc/edit?usp=sharing

Brett has been working on a few major projects since the beginning of the term. He has created the ability for users to edit rows on a table, as well as write notes for each row. Brett has also worked on writing libraries for form scrubbing and validation. When someone enters a form, the data gets cleaned up and ensured that all the values are in acceptable format.

Dylan has been working on the scrapping interface, so the users can logically delete items in the database.

Joseph has been hard at work on the facial recognition software and should hopefully have a working system implemented within the next few weeks.

Q. Weekly Blog Post 4/15 by Hayes, Brett

This week we have continued to work on our tasks and have been making our software more robust with less bugs. Brett has just about finished with validation and sanitization of the input forms. He has also been working on making sure all the tables on the web interface display nicely.

We have learned from our client that it may not be worth including the Intel logo in our work. This includes the poster and our project. So we are working on changing our poster and project to exclude the Intel logo. We are still allowed to use the word Intel in our work as long as it is used in the correct way.

R. Weekly Blog Post 5/9 by Hayes, Brett

We have forgotten the past few weeks to write a blog post, so this will be a larger post regarding what has been accomplished in the past few weeks.

April 16-23: We had a sanitizer validator for some of the tables, but not all of them. In this week we brought all of the tables currently in the system up to the same standard as the CPU table. The main reason why the other tables weren't at the same standard was because we were making constant changes to the structure of the tables, so once we came up with a solid standard for one table, we then made that standard for all tables. After getting all of the four tables in place, we added a fifth table for motherboards. These changes broght the tables into a finished state.

April 24-30: We changed the way the system handled database connections. Instead of a single connection, we made a pool of connections so that people can make database requests concurrently. This speeds up the system by removing the bottleneck of the system. We had also began further development on the displaying of item information on the kiosk. If the user scanned an item on the kiosk login page, they would get an alert with the item's information. It was only considered a temporary solution, and it only displayed CPU information. So we made a nicer solution for displaying information, and a user can scan any type of item and get the information. Finally, when developing for the barcode scanning portion, we wrote a program that simulates a barcode scan. It's a command-line program that, when given a serial number, will behave in the same way as our barcode scanner and type in each character of the serial number at a very fast rate. This has helped significantly with development, and allows us to test our code that requires a barcode scanner without actually using the barcode scanner.

May 1-7: We added a few new pages to the website. They both pertain to scrapping, or logically deleting, of items. The first page is the scrapped items page. It looks pretty much identical to the main page with tables. The only difference between the two pages is that the main page shows active items, and the new page shows scrapped items. The other page that was added is the "mass scrap" page. This allows a user to scan all the items they wish to scrap, and then once they are done scanning, can submit and set all those items to scrapped.

S. Weekly Blog Post 5/15 by Hayes, Brett

We have been putting in a lot of effort this past week in preparation for expo. All of us have been working very hard at making sure everything is finished on our project.

Brett has run through a number of bugs and features. He has fixed an issue with loging in from one interface and that session carrying over to the other interface. There is also a "quick" check in/out option on the kiosk login page. Admins can also edit the dropdown menus in the add item pages.

Dylan has been working on adding reservations to the system, so if an item is already checked out, it can then be reserved by another user.

Joseph is about done with the facial recognition, and should be up very soon. He has been having trouble with different SDKs and is finally about to have a working solution.

We are almost prepared for expo. We just have to make sure that our system can run between our machines, instead of relying on the internet in Kelley, since the internet won't be very reliable.

T. Weekly Blog Post 5/27 by Hayes, Brett

Due to expo being last Friday, we forgot to write our blog post last week. This blog post will contain the work accomplished within the past two weeks.

The week leading up to expo was filled with writing a lot of code and trying to make sure that everything was working before expo. We had finished a lot of tasks and bugs including the reservation of items, and facial recognition. The night

before expo we had to stay on campus through most of the night in order to finish the project and make sure it was ready. One of the reasons we were up so late was to make sure that our system would work without hiccups at expo. That included making sure that our two laptops were sharing a database, and that our website could work if we turned the internet off of our laptops. The other reason we were up so late is because we were working on getting facial recognition to work, and we had to work on it until the last minute. The Intel Realsense Camera was really difficult to work with, and it took a lot of hours to make sure that it worked correctly.

The day of expo went very well. None of us had gotten much sleep, but we were all excited to show off our project and we all felt proud of our work. We ended up being between two projects that were both interesting to the general public. To our left was a virtual reality implementation, and to our right was an RC car controlled remotely. Despite our project not being quite as flashy, we still were able to show it off to a lot of people, including some people representing some of the local industries. Our client, Scott was in town for the expo and he seemed happy with the work that we had put into this project over the months.

We haven't worked much after expo. There was a lecture on Wednesday and Kevin talked about the report and presentation. We have two weeks to finish both of these, but we will try to submit them sooner. This next week will be dedicated towards working on these, and after that, we will be finished with our capstone project.

IX. Project Documentation

A. How Does Your Project Work?

The project is broken into two interfaces. These are the kiosk, which is used for the checking in and checking out of items, and the web interface, which is used for visualizing the data. The kiosk should only be run on one machine. At Intel, they have a physical kiosk station where they will run the kiosk interface from. The web interface can be accessed from any personal device.

The kiosk interface is made up of two pages. The first is the kiosk login page. This page contains the login methods that our system supports. These are username and password combinations and facial recognition. The team at Intel has added a third option, which is RFID authentication. The kiosk login page also provides functionality to look up information of a particular item by scanning that items serial number. When scanning an item with the barcode scanner, it will detect the input regardless of where the cursor is currently. The information will be displayed on the right hand side, along with a button to save that item for checkout upon the next login. This saved for later storage is periodically cleared every 30 seconds to prevent unwanted items from being added to the cart in the case that someone saves an item for later but opts not to login. Finally, the last portion of the kiosk login page is the facial recognition interface. Users who have previously setup facial recognition will see their names in the table below the username and password fields. They can select their name and press the facial recognition button to attempt to login through facial recognition. It is important to look directly at the camera during the facial recognition process.

Once the user is authenticated, they will be introduced to the cart page. This page contains two tables, one for items to be checked in and one for items to be checked out. Items are scanned into the tables by scanning the item's barcode with the serial number scanner. Similar to the login page, the cursor's location is not important when scanning a barcode. After all items are scanned, simply pressing the submit button will complete the transaction.

The web interface, on the other hand, has many more pages. The first page a user will be greeted with upon entering the URL is the homepage. This page contains tables which list every item currently in the database. The different tables can be accessed by clicking on their corresponding tabs. Items can be searched in the table using a multi-column search. For example, one can search for all 4 core CPUs that also have a frequency of 2.5. Each entry in the table also has a button for adding notes. Clicking on the button creates a dropdown field for entering the note. If the user is logged in the admin privileges, each entry in the table will have yet another button. This button is for editing the items fields. Clicking on the button causes a modal to appear which contains a form filled with all of the items current values. Items can also be scrapped, or logically deleted, from this form.

The next page of the web interface is the add items pages. Clicking on the add items button in the navbar brings down a list of possible items that can be added to the database. Each add item page contains a form for adding the items. The forms are set up to sanitize and validate the data to make sure it is consistent and correct. Attempting to add an item incorrectly will produce an error below the field and the item will not be added to the database.

The next button on the navbar will bring the user to the reserve items page. This page is only accessible if the user is logged in. The page is similar to the homepage, except the table is only populated with items that are available for reservation. An item is only available for reservation if another user currently has that item checked out. A user may click on a third new button in an entry of the table to reserve the item. Once the item is reserved, the button will change to a different icon, and clicking the button again will inform the user when that item was reserved by them. Once the reserved item has been checked back in, that user will receive an email informing them that the item is now available for checkout.

The second to last button on the navbar will bring the user to the mass scrap page. This page is only accessible from users with admin privileges. This page is used for quickly scrapping a large quantity of items. Items are scanned into the table just like the cart page on the kiosk. Once the user has finished scanning the items he wishes to scrap, he/she must only press the submit button to scrap each item at the same time.

The final page of the web interface is the view scrapped page. This page also requires admin privileges. This page is for viewing the retired items that are no longer in use. It also is useful if an item has been scrapped by mistake, since the item can be found this table and un-scrapped by clicking the the edit item button.

B. How Does One Install the Project?

NOTE: We developed our project on Ubuntu, and our instructions are written assuming you are at least using a Debian version of Linux.

Our project is running on a Node.js server. Node.js can be downloaded from their website: https://nodejs.org. There is also a bash script in the root directory of our project to easily download and install node. Look for nodeinstall.sh. The latest version of Node should be fine. It will work with the stable or bleeding-edge version of Node.js.

After installing Node.js, navigate to the root folder and in the terminal type npm install. This will install any dependencies our program depends on. We use npm for all our packages. It comes installed with Node.js. For more information on npm, visit their website: https://www.npmjs.com/.

Our server uses a MySQL database. This can be installed either through a package manager with the command sudo apt-get install mysql-server if your using a debian distrobution of Linux, or directly from the binary on their website: https://www.mysql.com/downloads/. We use the MySQL Community Edition.

Facial recognition is done using OpenBR which needs to be installed on the server prior to facial recognition working. From the server, open a terminal window and type these commands:

Install GCC4.9.2

```
1  sudo apt-get update
2  sudo apt-get install build-essential
```

Install CMAKE3.0.2

```
1  sudo apt-get install cmake cmake-curses-gui
```

download OpenCV 2.4.11 from:
https://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.4.11/opencv-2.4.11.zip/download
After downloading, unzip opencv-2.4.11.zip

Download and Install OpenCV2.4.11

```
1  cd opencv-2.4.11
2  mkdir build
3  cd build
4  cmake -DCMAKE_BUILD_TYPE=Release ..
5  make -j4
6  sudo make install
7  cd ../..
8  rm -rf opencv-2.4.11*
```

Install QT5.4.1

```
1  sudo apt-get install qt5-default libqt5svg5-dev qtcreator
```

Get OpenBR

```
git clone https://github.com/biometrics/openbr.git
cd openbr
git checkout v1.1.0
git submodule init
git submodule update
```

Build OpenBR

```
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make -j4 NOTE: for faster install use more threads.
sudo make install
```

At this point you need to open the Qt Creator IDE using the command 'qtcreator &' NOTE: this requires a gui interface to be installed on the server. this is NOT a command line utility, however once OpenBR is installed the gui is no longer required. The gui we used was Ubuntu Desktop. This can be installed via package manager. In our case, this was "sudo apt-get install ubuntu-desktop".
From the Qt Creator "File" menu select "Open File or Project...".
Browse to your pre-existing build directory "openbr/build" then select "Next".
Select "Run CMake" then "Finish".
OpenBR should now be installed on the server and facial recognition should be functional.

C. How Does One Run the Program?

There are two main interfaces to the app.

1) The web interface
2) The kiosk interface.

The web interface can be found at the base url. To use the kiosk interface you must navigate to the /kiosk page. For example: the kiosk interface looks like: silicontracker.xyz/kiosk

We also have a second app that we have created to mock the credential login system at Intel. It's a very simplified version of their credential system (It's not actually how Intel deals with security. Our code is more of a placeholder). You will need to run this app if you plan on logging in as a user. open up a separate terminal window and go to the cred/ folder. There you will see a single file cred.js. Run this app with node cred.js.

The node.js app uses environment variables, and they are not included on the repo (no need for the world to know my passwords). In order to make the environment variables work, you must create a new file in the root directory called env.js. The file should look like the following, with your own settings to replace the ones here:

env.js

```

// dev  = development environment
// prod = production environment - needs to use https!
process.env.ENV       = 'dev';

// Database variables
process.env.DB_HOST     = 'localhost';
process.env.DB_USER     = 'tracker_user';
process.env.DB_PASSWORD   = 'dbpassword';
process.env.DB_SSL      = true;
process.env.DB        = 'tracker';

// Server app variables
process.env.APP_IP      = 'localhost';
process.env.APP_PORT    = 8080;
process.env.SSL_PASSPHRASE  = 'passphrase';

// Credential server variables
process.env.CRED_PORT   = 8082;
process.env.CRED_ADDR   = 'http://localhost:' + process.env.CRED_PORT;

// Session secret
process.env.SESSION_SECRET  = 'iamasecret';
```

```
24
25
26 // Email variables
27 process.env.EMAIL_USER = 'test.user@gmail.com';
28 process.env.EMAIL_PASSWORD = 'test123';
```

There are three things needed to get the database up and running:

1) Creating the database
2) Creating a user for the database
3) Populating the database

All of the database scripts we use are in the dbscripts folder. In a terminal window, navigate to that folder and login to your MySql client as root or as someone with sufficient privileges:

```
1 mysql -u root -p
```

After logging in, the first step is to create the database:

```
1 mysql> CREATE DATABASE tracker;
```

We called the database 'tracker' but you can rename it if you like. Just remember what you decided to call it. After creating, make sure to select the right database:

```
1 mysql> USE tracker;
```

Once the database is created, we need a user with the right privileges to be in charge of the database.

```
1 mysql> CREATE USER 'tracker_user'@'localhost' IDENTIFIED BY 'SomePassword';
```

You can change the name of the user if you like, but just remember what you named it. Next, we have to populate our newly-created database. The create scripts for the tables and stored procedures, as well as insert scripts for our sample data are in the script tracker.sql located in the dbscripts folder. We will be executing this script:

```
1 mysql> \. tracker.sql;
```

We will need to add a few permissions to our new user. It needs to select, update, and insert into the tables as well as execute stored procedures. This user will never be deleting rows, so it doesn't get that privilege.

```
1 mysql> GRANT SELECT ON tracker.* TO 'tracker_user'@'localhost';
2 mysql> GRANT INSERT ON tracker.* TO 'tracker_user'@'localhost';
3 mysql> GRANT UPDATE ON tracker.* TO 'tracker_user'@'localhost';
4 mysql> GRANT EXECUTE ON tracker.* TO 'tracker_user'@'localhost';
5 mysql> GRANT DELETE ON tracker.Checkout TO 'tracker_user'@'localhost';
6 mysql> FLUSH PRIVILEGES;
```

At this point, we should have our database completely set up. After setting up the server and downloading the dependencies, you should be able to start it up with npm start. Make sure you are in the root folder for the project before you run that command. You should be greeted with a message: 'Silicon Tracker Server listening at ¡ip address¿:¡port¿'. In your browser, navigate to that ip address. Don't forget the port! At this point you should be greeted with the Silicon Tracker website.

D. Are There Any Special Hardware, OS, or Runtime Requirements to Run Your Project?

There are not many specific requirements for the project, however there are a few. Facial recognition requires the Intel Realsense camera as it needs both the RGB and Depth cameras to function. Since the client side portions of the project are web based they will work with most any OS with a gui interface, though due to an issue in Linux where device names are truncated, it works better in Windows. For an unknown reason the web portions of this project do not function when using the Edge internet browser, but they work correctly in Firefox and Chrome.

The server side code was designed with a linux server in mind, and the code would require modification to work on a Windows server. The installation instructions and facial recognition script were also designed under the assumption that the server would be running Linux. The specific facial recognition algorithm used has a different set of installation and usage instructions for Windows and will not function as is on a non Linux server.

X. How Did You Learn New Technology?

A. What websites were helpful? (Listed in order of helpfulness.)

There were many helpful websites used during this project. Ranked in order of helpfulness they would be:

1) Stack Overflow
2) Github
3) Datatables.net
4) Intel.com

B. What, If Any, Reference Books Really Helped?

Only one book was used during this project as the majority of our information came from online, the book was Javascript: the Good Parts by Douglas Crockford, which was, as the title suggests, used when working with JavaScript for the server.

C. Were There Any People on Campus That Were Really Helpful?

There were two people who were helpful in completing the project. Professor McGrath and our client, Scott Oehrlein, both provided a great deal of help and suggestions for numerous parts of the project, but most of the help came when dealing with facial recognition and the Realsense SDK.

XI. What Did You Learn From This? (Dylan Camus)

A. What Technical Information Did You Learn?

I felt like I learned an enormous amount of technical knowledge over the course of this project. Coming into the project in the Fall I had very little web development experience. The only website I had ever built was a very simple checkout system using php for my intro to databases class.

The first major technology I learned was Javascript. Our project was written almost entirely in Javascript, and starting out I had almost no knowledge of Javascript. I learned how to write asynchronous code using callbacks, which our application made extensive use of. I learned how to structure a client server application in a model view controller format. I learned about the REST api and how to build a dynamic web application. I learned how to use tools like JQuery and Bootstrap to increase my productivity. I learned how to use git in a real project environment with a production branch and a development branch.

B. What Non-Technical Information Did You Learn?

I learned a lot about client interaction. We spoke over the phone with our client every week to talk about how the project was coming along. We also had a conference meeting with our client's coworkers about features that they liked, didn't like, and would like to see implemented. I also learned how to edit videos.

C. What Have You Learned About Project Work?

I learned that project work really helps productivity. I found that I would work much better when the entire team got together to work on the project at the same time because I could work out any difficulties I was having with the team. I also learned that working in a group is very helpful because of the diverse skillset each member brings to the table. Both of my group members possessed skills that I myself did not and it was useful to learn from them.

D. What Have You Learned About Project Management?

I learned that proper documentation is important. I felt that the first term was helpful because it gave me a good feel for the rest of the project. I also learned that keeping in contact with the client is extremely important for project management. Our client was very helpful at identifying bugs and giving feedback on the interface design.

E. What Have You Learned About Working in Teams?

I learned the importance of communication within a group. It is important to split up the work and make sure that everyone knows their own duties. There were times two people were working on the same task and productivity was wasted. It is also important to communicate to make sure everyone is on the same page and knows the direction we are going in. I learned that it is important to get the group's opinion before moving in a different direction.

F. If You Could Do It All over, What Would You Do Differently?

I would have put a greater emphasis on the facial recognition portion of the project. We did not value the facial recognition as all that important for most of the project and considered it as a cuttable feature. However, facial recognition was a required part of the project. By putting it off, we caused ourselves a lot of unnecessary stress when we scrambled to get the facial recognition portion of the project working.

XII. What Did You Learn From This? (Brett Hayes)

A. What Technical Information Did You Learn?

I learned a lot about system architecture and how components can be separate and interact with each other. Along with architecture, I felt like I got real experience on workflow, and making sure everything works as one cohesive entity. I learned to think about how changing one part of the system can affect other parts. This is especially true with an event-driven system, such as this one. Along with being event-driven, our system is completely asynchronous. It was a challenge learning how to think asynchronously, and to make sure everything works even if executed out of order.

B. What Non-Technical Information Did You Learn?

I learned a great deal about customer relations. We had a great client, and he was very knowledgeable in technology, so I felt we were lucky in that aspect. However as with any client, he didn't know exactly what he wanted from the project. He knew what core components he wanted to see in the system, but it took a lot of trying new things and seeing if our client liked the work we did or not. He also knew how the system was going to be used in production, so we had to rely on him for feedback in those areas. Overall, I learned much about how to effectively communicate ideas and software with a client.

C. What Have You Learned About Project Work?

There were times when the project was near completion, and I realized that we should have taken another design or chosen another piece of technology for a component in the system. I feel like it is okay to come up with these discoveries. I wouldn't have known that our choice in technology wasn't ideal until it was too late anyways. With that, I've also learned that code can be messy, and it's not always an option to go back and refactor the entire project.

D. What Have You Learned About Project Management?

I have learned that task lists and organization are key practices. Over the winter we didn't keep a strong task list and we stressed about how much work needed to be done. At the beginning of the spring, I came up with a task list for everything that needed to be done in order to push the software to production. We tracked requirements, user stories, task assignments, the status of the task, and estimated hours. It took out a lot of guessing to how much work was left to finish the project.

E. What Have You Learned About Working in Teams?

I have learned that communication is a big factor. It really helps when each member of a team communicates what task they are currently working on, and where they are on the project. This can lead to receiving help if one of the members is stuck, and making sure that two people aren't trying to accomplish the same task.

F. If You Could Do It All over, What Would You Do Differently?

There was some additional work that the client requested but wasn't in the original requirements document. If I could go back and do it all over, I would have made sure that the work in the requirements document was completed first before attempting any additional work. We ended up completing everything, so it all worked out in the end, but it did add a lot of stress and hours to the project.

XIII. What Did You Learn From This? (Joseph Cronise)

A. What Technical Information Did You Learn?

I learned a lot of technical information during the course of this project, ranging from new languages to new APIs. While, in the end, the project did not make use of the RealSense SDK, I learned a great deal about it while I was trying to make it work for our project. This also led me to learning C# as many of the Realsense samples were implemented in C# and the SDK had complete functionality with that language. I also expanded my understanding of javascript, previously I had only used it for websites, but during the project I had to use it to write server side scripts for the facial recognition system. I also learned how to work with bootstrap and the datatables API for the website as well as how to integrate webcams with a website.

B. What Non-Technical Information Did You Learn?

During this project I learned a lot about what I think of as real world coding. Instead of the specific instructions you get when taking a class, you are instead given a goal and, with some exceptions, it is left up to you as to how to reach that goal. Working with an actual client gave me a greater understanding of it works to create a real product that is going to be used, and it also gave me an understanding of just how easy it is for feature creep to occur.

C. What Have You Learned About Project Work?

During this project I learned a lot about project work and how it differs from the typical assignments given out in classes. I learned how easily project creep can occur, and how important it is to keep a clear set of goals from project start to finish. This project helped to give me a sense of how coding in a work environment might go and how multiple people can work together on different parts of a larger project and still work to create a cohesive project without stepping all over each others code in the process.

D. What Have You Learned About Project Management?

I wouldn't say that I learned a lot of new information about project management during this project. Coming from a business background I had already taken many of the project management courses and new a lot about what to expect going into a project of this nature. That said, I did learn just how important it is to keep a task list and keep it updated, and attempting to create the gantt chart for this paper taught me the importance of keeping clear records of who did what task, when they started, and when they finished.

E. What Have You Learned About Working in Teams?

By working as part of a team, I feel that we were able to create a product far better than anything each of us could have created individually. I learned that, by working as a team, we were able to leverage each person's individual skills and help everyone on their tasks as needed. Even something as simple as being able to bounce ideas off my teammates was incredibly helpful when it came to completing my tasks, and I would not have been able to implement the facial recognition system as well as it was without the help of my team.

F. If You Could Do It All over, What Would You Do Differently?

If I was starting over from scratch I would do a few things differently. I would have done much more research into the RealSense's capabilities earlier in the project cycle. This would have spared me a great deal of headache when it came time to make use of the SDK to write the facial recognition code, only to realise that the SDK has only partial javascript support, and no support for running its algorithms server side while the camera is client side. I would have written our kiosk in C# and made it an executable instead of creating it as a web interface. Had I done this I would have been able to make use of most of the RealSenses SDK, however I would have still had to implement my own database for it as the two functions required to save and load the recognition database within the SDk are non functioning.

XIV. Appendix 1: Essential Code Listings

App.js is the main entry point into the server. It is responsible for the main setup of the system, and it calls other files to help with the setup.

app.js

```
/*
 * app.js
 *
 * The main server code. Everything server-side is handled
 * by this app. This code is broken down into a few components.
 * There is the setup stage, execute stage, and router handling.
 */

/*
 * The setup stage
 */

var express = require('express');
var app = express();
var mysql = require('mysql');
var http = require('http')
var fs = require('fs');
var session = require('express-session');
var FileStore = require('session-file-store')(session);
var bodyParser = require('body-parser');
var nodemailer = require('nodemailer');
var schedule = require('node-schedule');
require('./app/templates')();

// Loading environment variables
require('./env.js');
process.env.ROOT_DIR = __dirname;
```

```
28
29 // Email templates
30 require('./app/templates.js')();
31
32 // Setting up the database
33 var db_options = {
34   host: process.env.DB_HOST || 'localhost',
35   user: process.env.DB_USER,
36   password: process.env.DB_PASSWORD,
37   database: process.env.DB || 'tracker'
38 };
39
40 if (process.env.DB_SSL === true) {
41   db_options.ssl = {
42     ca: fs.readFileSync(process.env.ROOT_DIR + '/ssl/silicontracker_xyz.crt')
43   }
44 }
45
46 var pool;
47 // handleConnection()
48 // When the database/server connection is lost due to connection issues,
49 // the system will attempt to reconnect.
50 function handleConnection() {
51   pool = mysql.createPool(db_options);
52   pool.getConnection(function(err, connection) {
53     if(err) {
54         console.log('error when connecting to db:', err);
55         setTimeout(handleConnection, 2000);
56       }  else {
57         console.log('Mysql pool connection established.');
58       }
59   });
60   pool.on('error', function(err) {
61     if(err.code === 'PROTOCOL_CONNECTION_LOST') {
62         handleConnection();
63       } else {
64         throw err;
65       }
66   });
67 }
68
69 handleConnection();
70
71 // Setting up Session environment
72
73 var sessOptions = {
74   name: 'tracker-cookie',
75   store: new FileStore,
76   secret: process.env.SESSION_SECRET,
77   cookie: {},
78   resave: false,
79   saveUninitialized: false,
80   name: 'my.tracker.sid'
81 };
82 if (process.env.ENV === 'prod') {
83   app.enable('trust proxy');
84   sessOptions.cookie.secure = true;
85 }
86 app.use(session(sessOptions));
87 app.use(bodyParser.urlencoded({ extended: true, parameterLimit: 10000, limit: '50mb'}));
88 app.use(express.static('public')); // All client-side code will be handled in the public folder.
89
90 /*
91  *The execute stage
92  */
93
94 // Activating the server
95 var ip = process.env.APP_IP || 'localhost';
96 var port = process.env.APP_PORT || 8080;
97
98 var server = http.createServer(app);
99 server.listen(port, ip, function(){
100   console.log('Silicon Tracker Server listening at ' + ip + ':' + port);
101 });
102
```

```
/*
 * Router Handling
 */

// Load the routes for the web data
require('./app/routes/data')(app, pool);

// Load the routes for the web pages
require('./app/routes/web')(app, pool);

// Load the routes for the kiosk
require('./app/routes/kiosk')(app, pool);

// Send emails for overdue items
var j = schedule.scheduleJob('00 00 * * 0', function() {
  var item_serial = [];
  var item_type = [];
  var days = [];
  var addr, first_name, last_name;
  pool.getConnection(function(err, conn) {
    conn.query("CALL get_overdue_owner();",
      function(error, results, fields) {
        if(error) {
          throw error;
        }
        var owner = results[0];
        for(var i in owner) {
          conn.query("CALL get_checkout_summary("+owner[i].wwid+","+owner[i].overdue_item_email_setting+");",
            function(error, results, fields) {
              if(error) {
                throw error;
              }
              var total_finished = 0;
              for(var j in results[0]) {
                item_serial[j] = results[0][j].serial_num;
                item_type[j] = results[0][j].item_type;
                days[j] = results[0][j].days;
                total_finished++;
                if(total_finished == results[0].length) {
                  addr = results[0][0].email_address;
                  first_name = results[0][0].first_name;
                  last_name = results[0][0].last_name;
                  console.log("Sending reminder email to "+addr+"...");
                  reminderTemplate(addr, first_name, last_name, item_serial, item_type, days);
                }
              }
            });
        }
      });
  });
});
```

Since we implemented our system to use the REST Api, we have many routes setup for different functionalites. For example, this route gets all the CPU silicon that has not been scrapped. It returns JSON back to the client, so the client can display the information in a table.

data.js

```
  app.get('/data/cpu', function(req, res) {
    var jsonToSend = {};
    pool.getConnection(function(err, conn) {
      conn.query("CALL get_cpu();", function(error, results, fields){
        if(error) {
          throw error;
        }
        conn.release();

        // We send admin stats for the table because there are admin-specific
        // elements to the table.
        if (isLoggedIn(req.session) && req.session.web.wwid) {
          jsonToSend.is_admin = req.session.web.is_admin;
        } else {
          jsonToSend.is_admin = 0;
        }
```

```
      var a = [];
      var user_name = null;
      for (var i in results[0]) {
        if(results[0][i].first_name == undefined || results[0][i].last_name == undefined) {
          user_name = null;
        } else {
          user_name = results[0][i].first_name + " " + results[0][i].last_name;
        }
        a.push(new models.CPU(results[0][i].serial_num, results[0][i].spec,
          results[0][i].mm, results[0][i].frequency, results[0][i].stepping,
          results[0][i].llc, results[0][i].cores, results[0][i].codename,
          results[0][i].cpu_class, results[0][i].external_name, results[0][i].architecture,
          results[0][i].user, results[0][i].checked_in, results[0][i].notes,
          results[0][i].scrapped, user_name));
      }
      jsonToSend.items = a;
      res.json(jsonToSend);
    });
  });
});
```

As long as there is a need to talk to the server, there is a route that handles it. Here we can see some more basic routes that handle adding and updating items.

web.js

```
app.post('/update/cpu', scrubAndValidate('CPU'), function(req, res) {
  pool.getConnection(function(err, conn) {
    conn.query("CALL update_cpu('"+req.body.serial_num+"','"
      +req.body.spec+"','"+req.body.mm+"','"
      +req.body.frequency+"','"+req.body.stepping+"','"
      +req.body.llc+"','"+req.body.cores+"','"
      +req.body.codename+"','"+req.body.cpu_class+"','"
      +req.body.external_name+"','"+req.body.architecture+"','"
      +req.body.notes+"','"+req.body.scrapped+"');",
    function(error, results, fields){
      if(error) {
        throw error;
      }
      conn.release();
    });
  });
  res.status(200).send(req.body);
});
```

```
app.post('/add/cpu', [scrubAndValidate('CPU'), checkSerials], function(req, res) {
  var stringifySerials = req.body.serial_num.toString();
  pool.getConnection(function(err, conn) {
    conn.query("CALL put_cpu('"+stringifySerials+"','"
      +req.body.spec+"','"+req.body.mm+"','"
      +req.body.frequency+"','"+req.body.stepping+"','"
      +req.body.llc+"','"+req.body.cores+"','"
      +req.body.codename+"','"+req.body.cpu_class+"','"
      +req.body.external_name+"','"+req.body.architecture+"','"
      +req.body.notes+"');",
      function(error, results, fields){
        if(error) {
          throw error;
        }
    });

    conn.release();
    res.status(200).send(req.body);
  });
});
```

If the user wants to login to our system, they will need to go through this route. Note that this is an authentication system that we created, and will look differently for Intel and their security system.

web.js

```
/*
 * Middleware for logging into the web interface. We need a few things done in sequential order.
 *
 * 1. Go to Intel's Active Directory Server, get WWID back.
```

```
   * 2. Check our database for user information using the WWID as a key.
   *
   * After that, we have the user's name and wwid stored in session, and can allow them into the cart.
   */
  app.use('/login/login', function(req, res, next) {
    request.post({url: process.env.CRED_ADDR, form: {'username': req.body.username, 'pass': req.body.pass
        }},
      function(error, response, body) {
        if (error) {
          console.log(error);
        }

        // Checking if the user exists in AD.
        if (body !== '') {
          req.session.web = new models.SessionUser();
          // We have a user in the AD system. Parse out the wwid.
          req.session.web.wwid = body;
          next();
        } else {
          // No wwid found, erase current session and create new session for user.
          res.redirect('/login');
        }
      });
  }, function(req,res,next) {
    // We know at this point we have a wwid, so let's try to get the user from our DB.
    pool.getConnection(function(err, conn) {
      conn.query('CALL get_user_from_wwid('+req.session.web.wwid+')', function(error, results, fields) {
          if (error) {
            throw error;
          }
          conn.release();

          if (results[0].length === 1) {
            req.session.web.last_name = results[0][0].last_name;
            req.session.web.first_name = results[0][0].first_name;
            req.session.web.is_admin = results[0][0].is_admin;
            req.session.web.loggedIn = true;
            req.session.save();
          } else {
            // Got no results
            res.redirect('/login');
          }
          next();
        });
    });
  });

  app.post('/login/login', function(req, res) {
    // If we made it this far through the middleware,
    // It must mean success. Send them to the cart!

    //TEST LOGS
    if (process.env.ENV === 'dev') {
      console.log(req.session.web.wwid);
      console.log(req.session.web.last_name);
      console.log(req.session.web.first_name);
    }

    res.redirect('/');
  });

};

function enforceLogin(req, res, next) {
  if(req.session.hasOwnProperty("web")) {
    if (req.session.web.loggedIn) {
      next();
    } else {
      res.redirect('/login');
    }
  } else {
    res.redirect('/login');
  }
}
```

```
function enforceAdminLogin(req, res, next) {
  if(req.session.hasOwnProperty("web")) {
    if (req.session.web.loggedIn && req.session.web.is_admin) {
      next();
    } else {
      res.redirect('/login');
    }
  } else {
    res.redirect('/login');
  }
}
```

Here is a route that handles many things, but the important thing to note is that it is called when a user submits a transaction after scanning items to be checked in or checked out.

kiosk.js

```
  app.post('/kiosk/submit', enforceLogin, function(req, res) {
    var item_type = [];
    var status = [];
    var items = [];
    var wwid = req.session.kiosk.wwid;
    var serial_nums = req.body.data;
    var total_finished = 0;
    var their_wwid = [];
    var their_addr = [];
    var their_first_name_reservation = [];
    var their_last_name_reservation = [];
    var addr, setting, first_name, last_name, date, user, their_last_name,
    their_first_name, the_date, my_first_name, my_last_name, my_first_name_reservation,
    my_last_name_reservation;
    if(req.body.hasOwnProperty("data")) {
      for(var i=0; i < serial_nums.length; i++) {
        pool.getConnection(function(i, err, conn) {
          conn.query("CALL scan_item("+ wwid +",'"+serial_nums[i]+"');",
            function(error, results, fields) {
              if(error) {
                throw error;
              }
              items[i] = {};
              items[i].serial_num = serial_nums[i];
              items[i].item_type = results[0][0].item_type;
              items[i].status = results[0][0].checked_in;
              user = results[0][0].user;
              total_finished++;

              if(items[i].status === 1) {
                conn.query("CALL get_reservation_id('"+items[i].serial_num+"');",
                  function(error, results, fields) {
                    for(var j=0; j < results[0].length; j++) {
                      if(error) {
                        throw error;
                      }
                      their_wwid[j] = results[0][j].waiting_user;

                      // get the contact info of that person
                      (function(j) {
                        conn.query("CALL get_addr("+their_wwid[j]+");",
                          function(error, results, fields) {
                            if(error) {
                              throw error;
                            }
                            their_addr[j] = results[0][0].email_address;
                            their_last_name_reservation[j] = results[0][0].last_name;
                            their_first_name_reservation[j] = results[0][0].first_name;
                            the_date = results[0][0].order_date;
                            // get the name of the person checking the item in
                            conn.query("CALL get_addr("+wwid+");",
                              function(error, results, fields) {
                                if(error) {
                                  throw error;
                                }
                                my_first_name_reservation = results[0][0].first_name;
                                my_last_name_reservation = results[0][0].last_name;

                                // send the email
```

```
                                console.log("Sending reservation email to "+their_addr+"...");
                                reservationTemplate(their_addr[j], their_first_name_reservation[j],
                                    their_last_name_reservation[j],
                                  my_first_name_reservation, my_last_name_reservation, items[i].
                                      serial_num, items[i].item_type, the_date);
                                // delete the reservation from the reservation table
                                conn.query("CALL delete_reservation('"+items[i].serial_num+"','"+
                                    their_wwid[j]+"');",
                                  function(error, results, fields) {
                                    if(error) {
                                      throw error;
                                    }
                                  });
                              });
                          });
                      })(j);
                    }
                  });
              }

              if(user != null && user != wwid) {
                conn.query("CALL get_addr("+user+");",
                  function(error, results, fields) {
                    if(error) {
                      throw error;
                    }

                    addr = results[0][0].email_address;
                    their_last_name = results[0][0].last_name;
                    their_first_name = results[0][0].first_name;
                    date = results[0][0].order_date;

                    conn.query("CALL get_addr("+wwid+");",
                      function(error, results, fields) {
                        if(error) {
                          throw error;
                        }
                        my_first_name = results[0][0].first_name;
                        my_last_name = results[0][0].last_name;
                        if(process.env.ENV === 'dev') {
                          console.log("Sending checkin email to "+addr+"...");
                        }
                        checkinTemplate(addr, their_first_name, their_last_name, my_first_name,
                            my_last_name, serial_nums[i], item_type[i], status[i], date);
                      });
                  });
              }

              if(total_finished === serial_nums.length) {
                conn.query("CALL get_addr("+wwid+");",
                  function(error, results, fields) {
                    if(error) {
                      throw error;
                    }

                    addr = results[0][0].email_address;
                    setting = results[0][0].cart_summary_email_setting;
                    last_name = results[0][0].last_name;
                    first_name = results[0][0].first_name;
                    date = results[0][0].order_date;

                    if(setting === 1) {
                      if (process.env.ENV === 'dev') {
                        console.log("Sending cart summary email to "+addr+"...");
                      }
                      cartTemplate(addr, first_name, last_name, serial_nums, item_type, status, date);
                    }
                  });
                conn.release();
                req.session.kiosk.loggedIn = false;
                req.session.saveForLater = [];
                res.status(200).json(items);
              }
            });
        }.bind(pool, i));
```

```
      }
    }
  });
```

Here is the code that sends the list of serial numbers from the client side to be checked in or out.

cart.js

```
// Submit serial numbers to the server.
function submitData(data) {
  $.post('/kiosk/submit', {data})
    .done(function(returnData) {
      // Success! display the modal.
      addModalRows(returnData);
      $('#SuccessModal').modal();

      val_array = [];

      // After 5 seconds, close the modal.
      setTimeout(function() {
        $('#SuccessModal').modal('hide');
      }, 5000);
    });
}
```

Depending on your love for JavaScript, here is some fun code that handles the facial recognition on the client. After taking images on the client-side, the system sends those images to the server to be saved to the database.

facial.js

```

// Using our custom attach method to accept camera id's.
Webcam.attach = newAttach;


function hasGetUserMedia() {
  return !!(navigator.getUserMedia || navigator.webkitGetUserMedia ||
            navigator.mozGetUserMedia || navigator.msGetUserMedia);
}
if (!hasGetUserMedia()) {
  alert('Warning: The webcam may not work for this browser.');
}

// http://jsfiddle.net/xLF38/818/
// Gets the average RGB values of the imgage provided.
function getAverageRGB(imgEl) {
  var blockSize = 5, // only visit every 5 pixels
    defaultRGB = {r:0,g:0,b:0}, // for non-supporting envs
    canvas = document.createElement('canvas'),
    context = canvas.getContext && canvas.getContext('2d'),
    data, width, height,
    i = -4,
    length,
    rgb = {r:0,g:0,b:0},
    count = 0;

    if (!context) {
      return defaultRGB;
    }

    height = canvas.height = imgEl.naturalHeight || imgEl.offsetHeight || imgEl.height;
    width = canvas.width = imgEl.naturalWidth || imgEl.offsetWidth || imgEl.width;

    context.drawImage(imgEl, 0, 0);

    try {
      data = context.getImageData(0, 0, width, height);
    } catch(e) {
      /* security error, img on diff domain */alert('x');
      return defaultRGB;
    }

    length = data.data.length;

    while ( (i += blockSize * 4) < length ) {
      ++count;
```

```
      rgb.r += data.data[i];
      rgb.g += data.data[i+1];
      rgb.b += data.data[i+2];
    }

    // ~~ used to floor values
    rgb.r = ~~(rgb.r/count);
    rgb.g = ~~(rgb.g/count);
    rgb.b = ~~(rgb.b/count);

    return rgb;
}

$(document).ready(function() {
  var depthCamId, rgbCamId;
  var camIds = [];

  var OSName="Unknown OS";
  if (navigator.appVersion.indexOf("Win")!=-1) { OSName="Windows" };
  if (navigator.appVersion.indexOf("Linux")!=-1) { OSName="Linux" };

  // Webcam setup
  Webcam.set({
    width: 320,
    height: 240,
    image_format: 'jpeg',
    jpeg_quality: 90
  });

  $('#SuccessModal').on('hidden.bs.modal', function(e) {
    location.reload();
  });

  // Linux cuts off the ends of the camera names, because they are too long.
  // The end of the camera names determine the differences between the cameras.
  // This makes it so we have to determine ourselves which camera is being run.
  // We do an analysis on the images after the images are taken.
  if (OSName === 'Linux') {
    // Find all the cameras
    navigator.mediaDevices.enumerateDevices()
    .then(function(devices) {
      for (var i = 0; i != devices.length; ++i) {
        var device = devices[i];
        if (device.kind === 'videoinput' && device.label.includes('RealSense')) {
          camIds.push(device.deviceId);
        }
      }
      // Attach the first camera.
      Webcam.attach('#my_camera0', camIds[0]);
      $('#snapshot').click(function() {
        var imagesToSend = [];
        // take snapshot and get image data from first camera
        Webcam.snap( function(data_uri0) {
          $('#results0').html(
            '<img id="myimg0" src="'+data_uri0+'"/>');


          // Attach second camera
          Webcam.attach('#my_camera1', camIds[1]);
          setTimeout(function() {
            // take snapshot of second camera
            Webcam.snap( function(data_uri1) {

              $('#results1').html(
                '<img id="myimg1" src="'+data_uri1+'"/>');

              // Since we don't know which image is which, we verify using
              // getting the average colors. The one that has way more green
              // than red or blue is the depth camera.
              var rgb = getAverageRGB(document.getElementById('myimg0'));
              if (rgb.g > rgb.b+100 && rgb.g > rgb.r+100) {
                // got depthcam
                imagesToSend.push({type: 'depth', uri: data_uri0});
                imagesToSend.push({type: 'rgb', uri: data_uri1});
              } else {
```

```
                // got rgbcam
                imagesToSend.push({type: 'rgb', uri: data_uri0});
                imagesToSend.push({type: 'depth', uri: data_uri1});
              }

              // after we have our images, send them to the server
              if (window.location.pathname === '/settings/facial-setup' ||
                  window.location.pathname === '/settings/facial-setup/') {
                $.post('/web/image', {images: imagesToSend})
                  .done(function() {
                    $('#SuccessModal').modal();

                    // After 5 seconds, close the modal.
                    setTimeout(function() {
                      $('#SuccessModal').modal('hide');
                    }, 5000);
                  });
              }

              if (window.location.pathname === '/kiosk' ||
                  window.location.pathname === '/kiosk/') {
                $.post('/kiosk/image', {images: imagesToSend, index: frTable.row('.row-select').data().
                    index})
                  .done(function() {
                    // being able to access the cart depends on whether or not the user
                    // is logged in via session. If they are logged in, they will be
                    // taken to the cart. Otherwise, they will attemt to redirect, but
                    // be redirected again to the login page.
                    window.location = '/cart';
                  });
              }
            });
          }, 1000);
        });
      });
    });
  }

  // Unlike Linux, Windows keeps the full names of the cameras.
  // This makes things simpler to determine each camera.
  if (OSName === 'Windows') {
    navigator.mediaDevices.enumerateDevices()
    .then(function(devices) {
      for (var i = 0; i != devices.length; ++i) {
        var device = devices[i];
        if (device.kind === 'videoinput' &&
            device.label.includes('RealSense') &&
            device.label.includes('Depth')) {
            // We have the depth camera
            depthCamId = device.deviceId;
        }
        if (device.kind === 'videoinput' &&
            device.label.includes('RealSense') &&
            device.label.includes('RGB')) {
            // We have the rgb camera
            rgbCamId = device.deviceId;
        }
      }
      // Attach the rgb camera.
      Webcam.attach('#my_camera0', rgbCamId);

      $('#snapshot').click(function() {
        // take snapshot and get image data from the rgb camera
        Webcam.snap( function(data_uri0) {
          $('#results0').html(
            '<img id="myimg0" src="'+data_uri0+'"/>');

          // Attach the depth camera
          Webcam.attach('#my_camera1', depthCamId);
          setTimeout(function() {
            // take snapshot of second camera
            Webcam.snap( function(data_uri1) {

              $('#results1').html(
                '<img id="myimg1" src="'+data_uri1+'"/>');
```

```

            // Grabbing all the images.
            var imagesToSend = [];
            imagesToSend.push({type: 'rgb', uri: data_uri0});
            imagesToSend.push({type: 'depth', uri: data_uri1});

            // after we have our images, send them to the server
            if (window.location.pathname === '/settings/facial-setup' ||
                window.location.pathname === '/settings/facial-setup/') {
              $.post('/web/image', {images: imagesToSend})
                .done(function() {
                  alert("images saved!");
                  location.reload();
                });
            }
            if (window.location.pathname === '/kiosk' ||
                window.location.pathname === '/kiosk/') {
              $.post('/kiosk/image', {images: imagesToSend, index: frTable.row('.row-select').data().
                  index})
                .done(function() {
                  // being able to access the cart depends on whether or not the user
                  // is logged in via session. If they are logged in, they will be
                  // taken to the cart. Otherwise, they will attemt to redirect, but
                  // be redirected again to the login page.
                  window.location = '/cart';
                });
            }
          });
        }, 1000);
      });
    });
  }
});
```

Some even more fun code than the facial recognition is the code that handles barcode scanners. It checks for key presses that happen very rapidly. That is how it determines if there was a barcode scan vs a human typing.

cart.js

```
// Barcode Scanner logic
$(window).keypress(function(e) {
  chars.push(String.fromCharCode(e.which));
  if (pressed == false) {
    setTimeout(function(){
    // If there are ten characters inserted before the timeout
    if (chars.length >= 10) {
      var barcode = chars.join("");
      barcode = barcode.replace("\r","");
      // There was an item scanned. Enter response code here.
      $.get('/serial/'+barcode, function(data) {
        if(data.err == 'Item Not Found') {
          $('#scan-error').html("<div>Item not found in database</div>");
          if($('#scan-error').is(":hidden")) {
            $('#scan-error').show();
          }
        } else if(data.scrapped === 'Scrapped' && data.checked_in === 'Checked In') {
          $('#scan-error').html("<div>Scrapped items cannot be checked out</div>");
          if($('#scan-error').is(":hidden")) {
            $('#scan-error').show();
          }
        } else if(arrayFind(val_array, function(n) {
          return n == data.serial_num;
        }) != -1) {
          $('#scan-error').html("<div>Item is already in the cart</div>");
          if($('#scan-error').is(":hidden")) {
            $('#scan-error').show();
          }
        } else {
          if($('#scan-error').is(":visible")) {
            $('#scan-error').hide();
          }
          if(data.checked_in === 'Checked In') {
            t_out.row.add([barcode]).draw();
          } else {
            t_in.row.add([barcode]).draw();
```

```
            }
            val_array.push(barcode);
          }
        });
      }
      chars = [];
      pressed = false;
      },250); // <-- this is the timeout in milliseconds
    }
    pressed = true;
  });
```

On our main page, there are a number of tables. Here is how we implemented the CPU table. It utilizes DataTables.js, a JQuery library for creating dynamic tables. As you can see from the code below, there are a lot of events and settings involved to get the tables just the way we want them.

table.js

```
$.get('/data/cpu', function(jsonData) {
  cpu_table = $('#cpu_table').DataTable({
  "data": jsonData.items,
  "columns" : [
    {
      "className": 'notes-control',
      "orderable": false,
      "data": null,
      "defaultContent": '',
      "visible": false
    },
    {
      "className": 'scrap-control',
      "orderable": false,
      "defaultContent": '',
      "data": "scrapped",
      "visible": false
    },
    {"data" : "serial_num"},
    {"data" : "spec"},
    {"data" : "mm"},
    {"data" : "frequency"},
    {"data" : "stepping"},
    {"data" : "llc"},
    {"data" : "cores"},
    {"data" : "codename"},
    {"data" : "cpu_class"},
    {"data" : "external_name"},
    {"data" : "architecture"},
    {"data" : "user_name"},
    {
      "defaultContent": '<button class="btn btn-link"><i class="fa fa-lg fa-file-o"></i></button>',
      "orderable": false,
      "className": "btn-notes",
    },
    {
      "defaultContent": '<button class="btn btn-link"><i class="fa fa-lg fa-pencil-square-o"></i></
          button>',
      "orderable": false,
      "className": "btn-edit",
      "visible": false,
    },
  ],
  "paging"      : true,
  "pagingType"  : "simple_numbers",
  "pageLength"  : 50,
  "fixedHeader" : {
       "header" : true,
       "footer" : false
  }
  });
  if (jsonData.is_admin === 1) {
    cpu_table.column(-1).visible(true);
  }
  // setting the column search bar width
  $('#cpuFilterCols th').each( function (idx) {
    var title = $(this).text();
```

```
      if (this.id !== 'colBtn') {
        $(this).html( '<input type="text" style="width: 95%;" placeholder="'+title+'" />' );
      }
    });
    // Apply the search
    $('#cpuFilterCols th').each(function (idx){
      // The plus 2 is needed because the first two columns are the
      // notes field (the child row) and scrapped field.
      var col = cpu_table.column(idx+2);
      $('input', this).on( 'keyup change', function () {
        col.search( this.value ).draw();
      });
    });

    // Add event listener for opening and closing details
    cpu_table.on('click', '.btn-notes', function () {
      var tr = $(this).closest('tr');
      var row = cpu_table.row(tr);

      if (row.child.isShown()) {
          // This row is already open - close it
          row.child.hide();
      } else {
        // Open this row
        row.child(format(row.data().notes)).show();

        // Setup form listener to send POST Ajax on submit.
        var childRow = $(tr).next();
        childRow.find('form').submit(function(e) {
          e.preventDefault();
          var newNotes = $(this).find('#notes').val();
          var dataToSend = {
            serial_num: row.data().serial_num,
            notes: newNotes
          };
          $.post('/update/notes', dataToSend, function(data, status, jqXHR) {
            if (status !== 'success') {
              alert('Error: Could not save notes.');
            } else {
              row.data().notes = newNotes;
              childRow.find('#save-status').text('Time saved: ' + moment().format('hh:mm:ss a'));
            }
          });
        });
      }
    });
    cpu_table.on('click', '.btn-edit', function() {
      var tr = $(this).closest('tr');
      var row = cpu_table.row(tr);
      cpu_data = {
        index: row.index(),
        serial_num: row.data().serial_num,
        spec: row.data().spec,
        mm: row.data().mm,
        frequency: row.data().frequency,
        stepping: row.data().stepping,
        llc: row.data().llc,
        cores: row.data().cores,
        codename: row.data().codename,
        cpu_class: row.data().cpu_class,
        external_name: row.data().external_name,
        architecture: row.data().architecture,
        notes: row.data().notes,
        scrapped: row.data().scrapped,
        user_name: row.data().user_name
      };
      $('#editCPUModal').modal('show');
    });

    // Placing the table in a horizontally scrollable box.
    // NOTE: Don't try using the scrollX DataTables option.
    // It messes with the column widths between the table header and body.
    var tableId = 'cpu_table';
    $('<div style="width: 100%; overflow: auto"></div>').append($('#' + tableId)).insertAfter($('#' +
        tableId + '_wrapper div').first());
```

```
});
```

```
$('#editCPUModal').on('show.bs.modal', function (event) {
  var modal = $(this);
  modal.find('#serial_num').val(cpu_data.serial_num);
  modal.find('#spec').val(cpu_data.spec);
  modal.find('#mm').val(cpu_data.mm);
  modal.find('#frequency').val(cpu_data.frequency);
  modal.find('#stepping').val(cpu_data.stepping);
  modal.find('#llc').val(cpu_data.llc);
  modal.find('#cores').val(cpu_data.cores);
  modal.find('#codename').val(cpu_data.codename);
  modal.find('#cpu_class').val(cpu_data.cpu_class);
  modal.find('#external_name').val(cpu_data.external_name);
  modal.find('#architecture').val(cpu_data.architecture);
  modal.find('#notes').val(cpu_data.notes);
  if(cpu_data.scrapped == 1) {
    modal.find('#scrap_cpu').prop('checked', true);
  } else {
    modal.find('#scrap_cpu').prop('checked', false);
  };
});
$('#editCPUSave').on('click', function() {
  var form = $(this).closest('.modal-content').find('form');
  cpu_data.spec = form.find('#spec').val();
  cpu_data.mm = form.find('#mm').val();
  cpu_data.frequency = form.find('#frequency').val();
  cpu_data.stepping = form.find('#stepping').val();
  cpu_data.llc = form.find('#llc').val();
  cpu_data.cores = form.find('#cores').val();
  cpu_data.codename = form.find('#codename').val();
  cpu_data.cpu_class = form.find('#cpu_class').val();
  cpu_data.external_name = form.find('#external_name').val();
  cpu_data.architecture = form.find('#architecture').val();
  cpu_data.notes = form.find('#notes').val();
  if(document.getElementById('scrap_cpu').checked) {
    cpu_data.scrapped = 1;
  } else {
    cpu_data.scrapped = 0;
  };

  // Getting the keys and values of all the fields in the form
  var obj = {};
  $.each($('#CPU').serializeArray(), function(_, kv) {
    obj[kv.name] = kv.value;
  });

  // Clearing all the old error messages
  $.each(obj, function(key, o) {
    var k = '#cpu_' + key + '_help';
    $(k).html('');
  });

  $.post('/update/cpu', cpu_data, function(data, status, jqXHR) {
    if(cpu_data.scrapped == 1) {
      //Remove scrapped item and update banner to reflect that
      cpu_table.row(cpu_data.index).remove();
      $.get('/data/stats', function(data) {
        $('#infoBanner').empty();
        $('#infoBanner').prepend('<div>Welcome ' + data.first_name + '</div>');
        $('#infoBanner').append('<span><strong>Total Items </strong>: ' +
                                data.num_active + ' active + ' +
                                data.num_scrapped + ' scrapped = ' +
                                data.num_total + '</span>');
      });
      cpu_table.draw();
    } else {
      cpu_table.row(cpu_data.index).data(cpu_data).draw();
    }
    $('#editCPUModal').modal('hide');
  })
  .fail(function(data) {
    // get the list of errors
    var errors = data.responseJSON;
```

```
73
74      // display the messages in the help divs
75      $.each(errors, function(key, messages) {
76        var k = '#cpu_' + key + '_help';
77
78        var repDOM = [];
79        $.each(messages, function(idx, m) {
80          repDOM.push('<span class="help-block"><p class="text-danger">'+m+'</p></span>')
81        });
82
83        $(k).append(repDOM);
84      });
85    });
86  });
87  $('#reserveCPUSave').on('click', function() {
88    var form = $(this).closest('.modal-content').find('form');
89    cpu_data.spec = form.find('#spec1').val();
90    cpu_data.mm = form.find('#mm').val();
91    cpu_data.frequency = form.find('#frequency').val();
92    cpu_data.stepping = form.find('#stepping').val();
93    cpu_data.llc = form.find('#llc').val();
94    cpu_data.cores = form.find('#cores').val();
95    cpu_data.codename = form.find('#codename').val();
96    cpu_data.cpu_class = form.find('#cpu_class').val();
97    cpu_data.external_name = form.find('#external_name').val();
98    cpu_data.architecture = form.find('#architecture').val();
99    cpu_data.notes = form.find('#notes').val();
100   if(document.getElementById('scrap_cpu').checked) {
101     cpu_data.scrapped = 1;
102   } else {
103     cpu_data.scrapped = 0;
104   };
105
106   // Getting the keys and values of all the fields in the form
107   var obj = {};
108   $.each($('#CPU').serializeArray(), function(_, kv) {
109     obj[kv.name] = kv.value;
110   });
111
112   // Clearing all the old error messages
113   $.each(obj, function(key, o) {
114     var k = '#cpu_' + key + '_help';
115     $(k).html('');
116   });
117
118   $.post('/update/cpu', cpu_data, function(data, status, jqXHR) {
119     if(cpu_data.scrapped == 1) {
120       //Remove scrapped item and update banner to reflect that
121       cpu_table.row(cpu_data.index).remove();
122       $.get('/data/stats', function(data) {
123         $('#infoBanner').empty();
124         $('#infoBanner').prepend('<div>Welcome ' + data.first_name + '</div>');
125         $('#infoBanner').append('<span><strong>Total Items </strong>: ' +
126                                 data.num_active + ' active + ' +
127                                 data.num_scrapped + ' scrapped = ' +
128                                 data.num_total + '</span>');
129       });
130       cpu_table.draw();
131     } else {
132       cpu_table.row(cpu_data.index).data(cpu_data).draw();
133     }
134     $('#editCPUModal').modal('hide');
135   })
136   .fail(function(data) {
137     // get the list of errors
138     var errors = data.responseJSON;
139
140     // display the messages in the help divs
141     $.each(errors, function(key, messages) {
142       var k = '#cpu_' + key + '_help';
143
144       var repDOM = [];
145       $.each(messages, function(idx, m) {
146         repDOM.push('<span class="help-block"><p class="text-danger">'+m+'</p></span>')
147       });
```

```
        $(k).append(repDOM);
      });
    });
  });
```

Below are the scubbers and validators for all the items. The scrubbers are in charge of converting the data into usable formats, and after the data has been scrubbed, it goes through the validator to make sure the input the user added was valid. If the input was not valid, the system will send the client an array of messages to let them how how to remedy the issues.

scrubbers.js

```
module.exports = {
  CPU: function(cpu) {
    return scrubCPU(cpu);
  },
  SSD: function(ssd) {
    return scrubSSD(ssd);
  },
  Memory: function(mem) {
    return scrubMemory(mem);
  },
  Flash: function(flash) {
    return scrubFlash(flash);
  },
  Board: function(board) {
    return scrubBoard(board);
  }
};

// Capitalize each word in a string
String.prototype.capitalize = function() {
    return this.replace(/(?:^|\s)\S/g, function(a) { return a.toUpperCase(); });
};

// Replace extra spaces between each word, along with leading and trailing whitespaces.
String.prototype.trimWords = function() {
  return this.replace(/\s+/g, " ").trim();
}

function scrubCPU(cpu) {
  // Serial Number
  // 1. Convert to uppercase
  // 2. Split into array based on newline as the delimeter
  // 3. Trim whitespace
  if (cpu.hasOwnProperty('serial_num')) {
    if (cpu.serial_num !== '') {
      cpu.serial_num = cpu.serial_num.toUpperCase().split(/\n/);
      for (var i = 0; i < cpu.serial_num.length; i++) {
        cpu.serial_num[i] = cpu.serial_num[i].trimWords();
      }

      for (var i = 0; i < cpu.serial_num.length; i++) {
        if (cpu.serial_num[i] === '') {
          cpu.serial_num.splice(i, 1);
          i--;
        }
      }
    } else {
      cpu.serial_num = [];
    }
  }

  // Spec
  // 1. Convert to uppercase
  // 2. Trim whitespace
  if (cpu.hasOwnProperty('spec')) {
    cpu.spec = cpu.spec.toUpperCase().trimWords();
  }

  // MM
  // 1. Trim whitespace
  if (cpu.hasOwnProperty('mm')) {
    cpu.mm = cpu.mm.trimWords();
  }
```

```
  // Frequency
  // 1. Remove trailing zeros from string by converting string to float
  //    (will automatically trim whitespace)
  // 2. Make sure it's in string format
  if (cpu.hasOwnProperty('frequency')) {
    cpu.frequency = parseFloat(cpu.frequency).toString();
  }

  // Stepping
  // 1. Convert to uppercase
  // 2. Trim whitespace
  if (cpu.hasOwnProperty('stepping')) {
    cpu.stepping = cpu.stepping.toUpperCase().trimWords();
  }

  // LLC
  // 1. Trim whitespace and trailing zeros
  if (cpu.hasOwnProperty('llc')) {
    cpu.llc = parseFloat(cpu.llc);
  }

  // Cores
  // 1. Trim whitespace
  // 2. Convert to int
  if (cpu.hasOwnProperty('cores')) {
    cpu.cores = parseInt(cpu.cores.trim());
  }

  // Codename
  // 1. Capitalize each word
  // 2. Trim whitespace
  if (cpu.hasOwnProperty('codename')) {
    cpu.codename = cpu.codename.capitalize().trimWords();
  }

  // CPU Class
  // 1. Capitalize each word
  // 2. Trim whitespace
  if (cpu.hasOwnProperty('cpu_class')) {
    cpu.cpu_class = cpu.cpu_class.capitalize().trimWords();
  }

  // External Name
  // 1. Capitalize just the first word, since the format may contain abbreviations and numbers.
  //    Some abbreviations may not want to be capitalized (e.g. i7) so we leave that input up to the user.
  // 2. Trim whitespace
  if (cpu.hasOwnProperty('external_name')) {
    cpu.external_name = cpu.external_name.replace(/(?:^|\s)\S/, function(a) { return a.toUpperCase(); });
    cpu.external_name = cpu.external_name.trimWords();
  }

  // Architecture
  // 1. Capitalize each word
  // 2. Trim whitespace
  if (cpu.hasOwnProperty('architecture')) {
    cpu.architecture = cpu.architecture.capitalize().trimWords();
  }

  // Notes
  // 1. Trim leading and trailing whitespace.
  //    (Don't need to trim between words. Let the user decide how to use notes).
  if (cpu.hasOwnProperty('notes')) {
    cpu.notes = cpu.notes.trim();
  }

  return cpu;
}

function scrubSSD(ssd) {
  // Serial Number
  // 1. Convert to uppercase
  // 2. Split into array based on newline as the delimeter
  // 3. Trim whitespace
```

```
  if (ssd.hasOwnProperty('serial_num')) {
    if (ssd.serial_num !== '') {
      ssd.serial_num = ssd.serial_num.toUpperCase().split(/\n/);
      for (var i = 0; i < ssd.serial_num.length; i++) {
        ssd.serial_num[i] = ssd.serial_num[i].trimWords();
      }
    } else {
      ssd.serial_num = [];
    }
  }

  // Manufacturer
  // 1. Capitalize each word
  // 2. Trim whitespace
  if (ssd.hasOwnProperty('manufacturer')) {
    ssd.manufacturer = ssd.manufacturer.capitalize().trimWords();
  }

  // Model
  // 1. Capitalize each word
  // 2. Trim whitespace
  if (ssd.hasOwnProperty('model')) {
    ssd.model = ssd.model.capitalize().trimWords();
  }

  // Capacity
  // 1. Trim whitespace
  // 2. Convert to int
  if (ssd.hasOwnProperty('capacity')) {
    ssd.capacity = parseInt(ssd.capacity.trim());
  }

  // Notes
  // 1. Trim leading and trailing whitespace.
  //    (Don't need to trim between words. Let the user decide how to use notes).
  if (ssd.hasOwnProperty('notes')) {
    ssd.notes = ssd.notes.trim();
  }

  return ssd;
}

function scrubMemory(mem) {
  // Serial Number
  // 1. Convert to uppercase
  // 2. Split into array based on newline as the delimeter
  // 3. Trim whitespace
  if (mem.hasOwnProperty('serial_num')) {
    if (mem.serial_num !== '') {
      mem.serial_num = mem.serial_num.toUpperCase().split(/\n/);
      for (var i = 0; i < mem.serial_num.length; i++) {
        mem.serial_num[i] = mem.serial_num[i].trimWords();
      }
    } else {
      mem.serial_num = [];
    }
  }

  // Manufacturer
  // 1. Capitalize each word
  // 2. Trim whitespace
  if (mem.hasOwnProperty('manufacturer')) {
    mem.manufacturer = mem.manufacturer.capitalize().trimWords();
  }

  // Physical Size
  // 1. Capitalize each word
  // 2. Trim whitespace
  if (mem.hasOwnProperty('physical_size')) {
    mem.physical_size = parseInt(mem.physical_size.trim());
  }

  // ECC
  // 1. Capitalize each word
  // 2. Trim whitespace
```

```
  if (mem.hasOwnProperty('ecc')) {
    mem.ecc = mem.ecc.capitalize().trimWords();
  }

  // Ranks
  // 1. Trim whitespace
  // 2. Convert to int
  if (mem.hasOwnProperty('ranks')) {
    mem.ranks = parseInt(mem.ranks.trim());
  }

  // Memory Type
  // 1. Convert to uppercase
  // 2. Trim whitespace
  if (mem.hasOwnProperty('memory_type')) {
    mem.memory_type = mem.memory_type.toUpperCase().trimWords();
  }

  // Capacity
  // 1. Trim whitespace
  // 2. Convert to int
  if (mem.hasOwnProperty('capacity')) {
    mem.capacity = parseInt(mem.capacity.trim());
  }

  // Speed
  // 1. Trim whitespace
  // 2. Convert to int
  if (mem.hasOwnProperty('speed')) {
    mem.speed = parseInt(mem.speed.trim());
  }

  // Notes
  // 1. Trim leading and trailing whitespace.
  //    (Don't need to trim between words. Let the user decide how to use notes).
  if (mem.hasOwnProperty('notes')) {
    mem.notes = mem.notes.trim();
  }

  return mem;
}

function scrubFlash(flash) {
  // Serial Number
  // 1. Convert to uppercase
  // 2. Split into array based on newline as the delimeter
  // 3. Trim whitespace
  if (flash.hasOwnProperty('serial_num')) {
    if (flash.serial_num !== '') {
      flash.serial_num = flash.serial_num.toUpperCase().split(/\n/);
      for (var i = 0; i < flash.serial_num.length; i++) {
        flash.serial_num[i] = flash.serial_num[i].trimWords();
      }
    } else {
      flash.serial_num = [];
    }
  }

  // Manufacturer
  // 1. Capitalize each word
  // 2. Trim whitespace
  if (flash.hasOwnProperty('manufacturer')) {
    flash.manufacturer = flash.manufacturer.capitalize().trimWords();
  }

  // Capacity
  // 1. Trim whitespace
  // 2. Convert to int
  if (flash.hasOwnProperty('capacity')) {
    flash.capacity = parseInt(flash.capacity.trim());
  }

  // Notes
  // 1. Trim leading and trailing whitespace.
  //    (Don't need to trim between words. Let the user decide how to use notes).
```

```
  if (flash.hasOwnProperty('notes')) {
    flash.notes = flash.notes.trim();
  }

  return flash;
}

function scrubBoard(board) {
  // Serial Number
  // 1. Convert to uppercase
  // 2. Split into array based on newline as the delimeter
  // 3. Trim whitespace
  if (board.hasOwnProperty('serial_num')) {
    if (board.serial_num !== '') {
      board.serial_num = board.serial_num.toUpperCase().split(/\n/);
      for (var i = 0; i < board.serial_num.length; i++) {
        board.serial_num[i] = board.serial_num[i].trimWords();
      }
    } else {
      board.serial_num = [];
    }
  }

  // FPGA
  // 1. Capitalize each word
  // 2. Trim whitespace
  if (board.hasOwnProperty('fpga')) {
    board.fpga = board.fpga.capitalize().trimWords();
  }

  // BIOS
  // 1. Trim whitespace
  if (board.hasOwnProperty('bios')) {
    board.bios = board.bios.trimWords();
  }

  // MAC Address
  // 1. Trim whitespace
  if (board.hasOwnProperty('mac')) {
    board.mac = board.mac.trim();
  }

  // Fab
  // 1. Capitalize each word
  // 2. Trim whitespace
  if (board.hasOwnProperty('fab')) {
    board.fab = board.fab.capitalize().trimWords();
  }

  // Notes
  // 1. Trim leading and trailing whitespace.
  //    (Don't need to trim between words. Let the user decide how to use notes).
  if (board.hasOwnProperty('notes')) {
    board.notes = board.notes.trim();
  }

  return board;
}
```

validators.js

```
// How to use the validator: validatejs.org

var validate = require('validate.js');
module.exports = {
  CPU: function(cpu) {
    var val = validate(cpu, CPUConstraints);

    // checking each serial_num number
    for (var i = 0; i < cpu.serial_num.length; i++) {
      var toValidate = {serial_num: cpu.serial_num[i]}
      var v = validate(toValidate, CPUSerialConstraints);
      if (v) {
        // checking if any errors at all (if undefined or null)
        if (val != null) {
          // checking if any errors yet for serial_num numbers
```

```
          if (val.hasOwnProperty('serial_num')) {
            // push each serial_num number error
            for (var j = 0; j < v.serial_num.length; j++) {
              val.serial_num.push(v.serial_num[j]);
            }
          } else {
            val.serial_num = v.serial_num;
          }
        } else {
          // if validate value is undefined, start new set of errors
          val = v;
        }
      }
    }

    return val;
  },
  SSD: function(ssd) {
    var val = validate(ssd, SSDConstraints);

    // checking each serial_num number
    for (var i = 0; i < ssd.serial_num.length; i++) {
      var toValidate = {serial_num: ssd.serial_num[i]}
      var v = validate(toValidate, SSDSerialConstraints);
      if (v) {
        // checking if any errors at all (if undefined or null)
        if (val != null) {
          // checking if any errors yet for serial_num numbers
          if (val.hasOwnProperty('serial_num')) {
            // push each serial_num number error
            for (var j = 0; j < v.serial_num.length; j++) {
              val.serial_num.push(v.serial_num[j]);
            }
          } else {
            val.serial_num = v.serial_num;
          }
        } else {
          // if validate value is undefined, start new set of errors
          val = v;
        }
      }
    }

    return val;
  },
  Memory: function(mem) {
    var val = validate(mem, MemoryConstraints);

    // checking each serial_num number
    for (var i = 0; i < mem.serial_num.length; i++) {
      var toValidate = {serial_num: mem.serial_num[i]}
      var v = validate(toValidate, MemorySerialConstraints);
      if (v) {
        // checking if any errors at all (if undefined or null)
        if (val != null) {
          // checking if any errors yet for serial_num numbers
          if (val.hasOwnProperty('serial_num')) {
            // push each serial_num number error
            for (var j = 0; j < v.serial_num.length; j++) {
              val.serial_num.push(v.serial_num[j]);
            }
          } else {
            val.serial_num = v.serial_num;
          }
        } else {
          // if validate value is undefined, start new set of errors
          val = v;
        }
      }
    }

    return val;
  },
  Flash: function(flash) {
    var val = validate(flash, FlashConstraints);
```

```
91
92      // checking each serial_num number
93      for (var i = 0; i < flash.serial_num.length; i++) {
94        var toValidate = {serial_num: flash.serial_num[i]}
95        var v = validate(toValidate, FlashSerialConstraints);
96        if (v) {
97          // checking if any errors at all (if undefined or null)
98          if (val != null) {
99            // checking if any errors yet for serial_num numbers
100           if (val.hasOwnProperty('serial_num')) {
101             // push each serial_num number error
102             for (var j = 0; j < v.serial_num.length; j++) {
103               val.serial_num.push(v.serial_num[j]);
104             }
105           } else {
106             val.serial_num = v.serial_num;
107           }
108         } else {
109           // if validate value is undefined, start new set of errors
110           val = v;
111         }
112       }
113     }
114
115     return val;
116   },
117   Board: function(board) {
118     var val = validate(board, BoardConstraints);
119
120     // checking each serial_num number
121     for (var i = 0; i < board.serial_num.length; i++) {
122       var toValidate = {serial_num: board.serial_num[i]}
123       var v = validate(toValidate, BoardSerialConstraints);
124       if (v) {
125         // checking if any errors at all (if undefined or null)
126         if (val != null) {
127           // checking if any errors yet for serial_num numbers
128           if (val.hasOwnProperty('serial_num')) {
129             // push each serial_num number error
130             for (var j = 0; j < v.serial_num.length; j++) {
131               val.serial_num.push(v.serial_num[j]);
132             }
133           } else {
134             val.serial_num = v.serial_num;
135           }
136         } else {
137           // if validate value is undefined, start new set of errors
138           val = v;
139         }
140       }
141     }
142
143     return val;
144   },
145 };
146
147 var attrNames = {
148   // CPU
149   serial_num:   'Serial Number',
150   spec:   'Spec',
151   mm:     'MM',
152   frequency:  'Frequency',
153   stepping:   'Stepping',
154   llc:    'LLC',
155   cores:  'Cores',
156   codename: 'Codename',
157   cpu_class:  'Class',
158   external_name:  'External Name',
159   architecture:   'Architecture',
160   notes:  'Notes',
161
162   // SSD
163   capacity:   'Capacity',
164   manufacturer: 'Manufacturer',
165   model:    'Model',
```

```
  // Memory
  physical_size:  'Physical Size',
  memory_type:    'Type',
  speed:      'Speed',
  ecc:        'ECC',
  ranks:      'Ranks',

  // Flash Drives

  // Boards
  fpga: 'FPGA',
  bios: 'BIOS',
  mac: 'MAC Address',
  fab: 'Fab',

  // Other
  greaterThan:      'greater than',
  greaterThanOrEqualTo:   'greater than or equal to',
  equalTo:        'equal to',
  lessThan:         'less than',
  lessThanOrEqualTo:    'less than or equal to'
};

// Override prettify to be just a key-value finder.
validate.prettify = function(str) {
  for (var key in attrNames) {
    if (str === key) {
      return attrNames[key];
    }
  }
  return str;
}

// Overriding the precense message.
validate.validators.presence.message = "is required";

var CPUSerialConstraints = {
  serial_num: {
    length: {
      maximum: 14,
      message: '^%{value} must be less than 14 characters in length.'
    },
    format: {
      pattern: /[a-zA-Z0-9]+/,
      message: '^%{value} must be alphanumeric (letters and numbers).'
    }
  }
}

var CPUConstraints = {
  //Serial Number
  serial_num: {
    presence: true
  },
  // Spec
  spec: {
    presence: true,
    length: {
      maximum: 5
    },
    format: {
      pattern: /[a-zA-Z0-9]+/,
      message: 'must be alphanumeric (letters and numbers).'
    }

  },
  // MM
  mm: {
    presence: true,
    numericality: {
      onlyInteger: true,
      greaterThan: 9999
    },
    length: {
```

```
      maximum: 7
    }
  },
  // Frequency
  frequency: {
    presence: true,
    numericality: {
      greaterThan: 0
    }
  },
  // Stepping
  stepping: {
    presence: true,
    length: {
      maximum: 6
    }
  },
  // LLC
  llc: {
    presence: true,
    numericality: {
      onlyInteger: true,
      greaterThan: 0
    }
  },
  // Cores
  cores: {
    presence: true,
    numericality: {
      onlyInteger: true,
      greaterThan: 0
    }
  },
  // Codename
  codename: {
    presence: true,
    length: {
      maximum: 25
    }
  },
  // CPU Class
  cpu_class: {
    presence: true,
    length: {
      maximum: 10
    }
  },
  // External Name
  external_name: {
    length: {
      maximum: 25
    }
  },
  // Architecture
  architecture: {
    presence: true,
    length: {
      maximum: 25
    }
  },
  // Notes
  notes: {

  }
};

var SSDSerialConstraints = {
  serial_num: {
    length: {
      maximum: 16,
      message: '^%{value} must be 16 characters or less in length.'
    },
    format: {
      pattern: /[a-zA-Z0-9]+/,
      message: '^%{value} must be alphanumeric (letters and numbers).'
```

```
    }
  }
}

var SSDConstraints = {
  // Serial Number
  serial_num: {
    presence: true
  },
  // Capacity
  capacity: {
    presence: true,
    numericality: {
      onlyInteger: true,
      greaterThan: 0
    }
  },
  // Manufacturer
  manufacturer: {
    presence: true,
    length: {
      maximum: 45
    }
  },
  // Model
  model: {
    presence: true,
    length: {
      maximum: 15
    }
  },
  // Notes
  notes: {

  }
};

var MemorySerialConstraints = {
  serial_num: {
    length: {
      maximum: 20,
      message: '^%{value} must be 20 characters or less in length.'
    },
    format: {
      pattern: /[a-zA-Z0-9]+/,
      message: '^%{value} must be alphanumeric (letters and numbers).'
    }
  }
};

var MemoryConstraints = {
  // Serial Number
  serial_num: {
    presence: true
  },
  // Manufacturer
  manufacturer: {
    presence: true,
    length: {
      maximum: 45
    }
  },
  // Physical Size
  physical_size: {
    presence: true,
    numericality: {
      onlyInteger: true,
      greaterThan: 0
    }
  },
  // ECC
  ecc: {
    presence: true,
    inclusion: {
      within: ['Yes', 'No']
```

```
    }
  },
  // Ranks
  ranks: {
    presence: true,
    numericality: {
      onlyInteger: true
    }
  },
  // Memory Type
  memory_type: {
    presence: true,
    length: {
      maximum: 12
    }
  },
  // Capacity
  capacity: {
    presence: true,
    numericality: {
      onlyInteger: true,
      greaterThan: 0
    }
  },
  // Speed
  speed: {
    presence: true,
    numericality: {
      greaterThan: 0
    }
  },
  // Notes
  notes: {

  }
};

var FlashSerialConstraints = {
  serial_num: {
    length: {
      maximum: 20,
      message: '^%{value} must be 20 characters or less in length.'
    },
    format: {
      pattern: /[a-zA-Z0-9]+/,
      message: '^%{value} must be alphanumeric (letters and numbers).'
    }
  }
};

var FlashConstraints = {
  // Serial Number
  serial_num: {
    presence: true
  },
  // Manufacturer
  manufacturer: {
    presence: true,
    length: {
      maximum: 45
    }
  },
  // Capacity
  capacity: {
    presence: true,
    numericality: {
      onlyInteger: true,
      greaterThan: 0
    }
  },
  // Notes
  notes: {

  }
};
```

```

var BoardSerialConstraints = {
  serial_num: {
    length: {
      maximum: 20,
      message: '^%{value} must be 16 characters or less in length.'
    },
    format: {
      pattern: /[a-zA-Z0-9]+/,
      message: '^%{value} must be alphanumeric (letters and numbers).'
    }
  }
}

var BoardConstraints = {
  // Serial Number
  serial_num: {
    presence: true
  },
  // FPGA
  fpga: {
    length: {
      maximum: 30
    }
  },
  // BIOS
  bios: {
    length: {
      maximum: 75
    }
  },
  // MAC Address
  mac: {
    length: {
      maximum: 17
    }
  },
  fab: {
    length: {
      maximum: 10
    }
  },
  // Notes
  notes: {
  }
};
```

XV. Appendix 2: Additional Photos

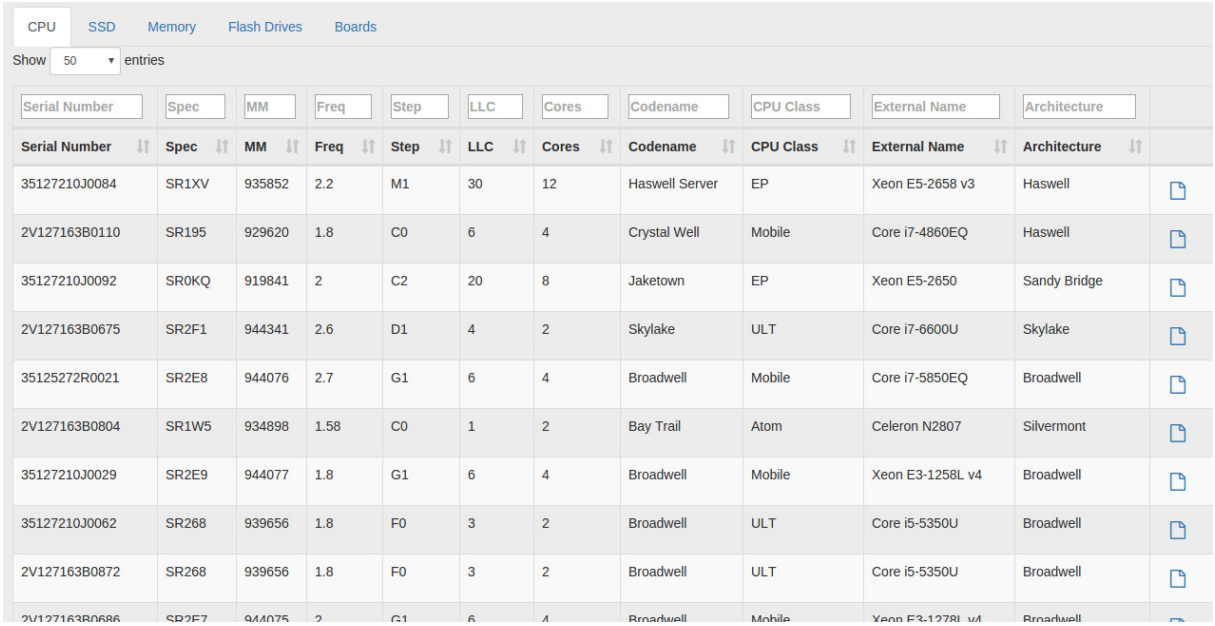


Fig. 1: The CPU table.

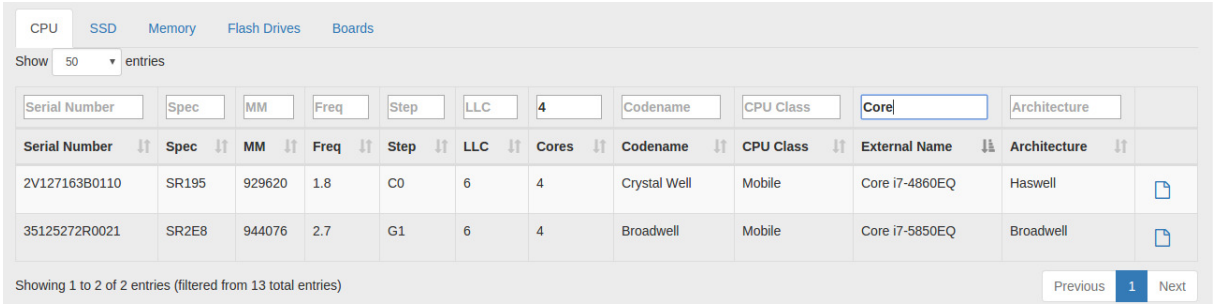


Fig. 2: Filtering table based on number of cores and external name.

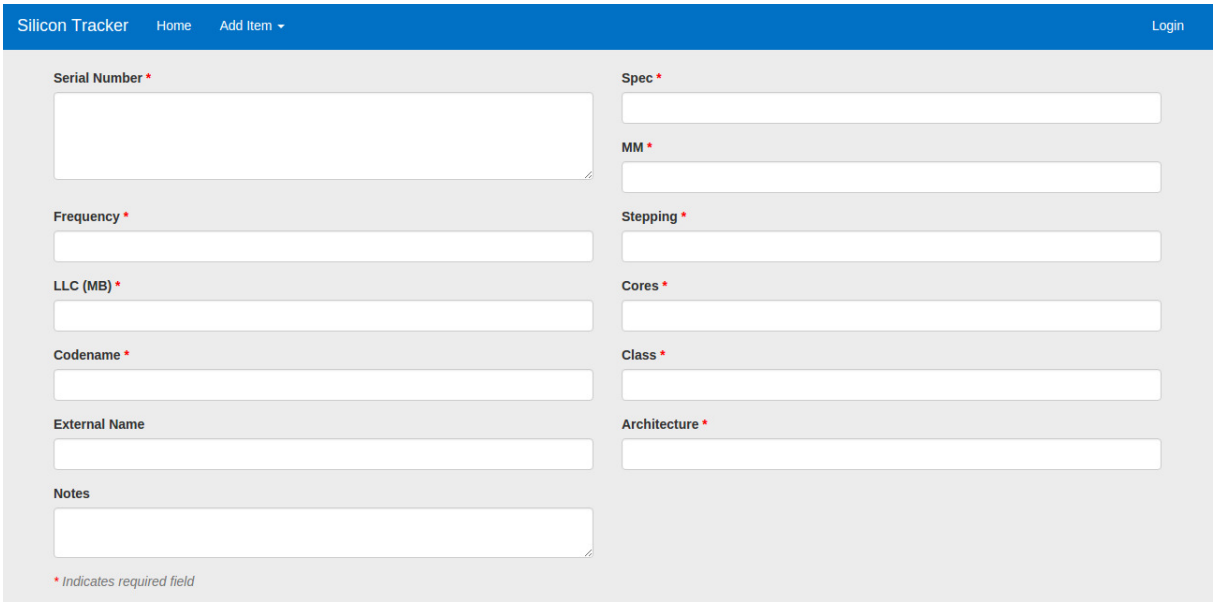


Fig. 3: The screen for adding a new CPU.

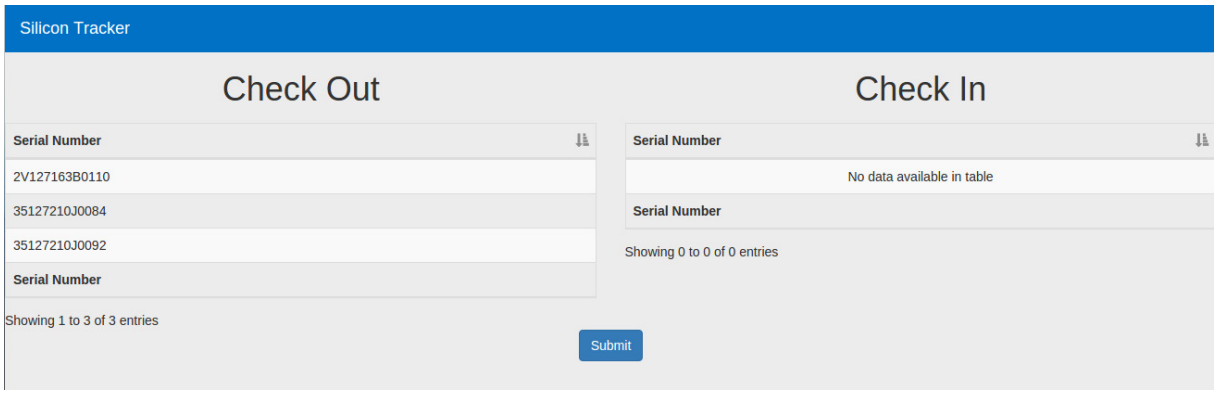


Fig. 4: The kiosk cart screen, with some items to check out.