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

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.

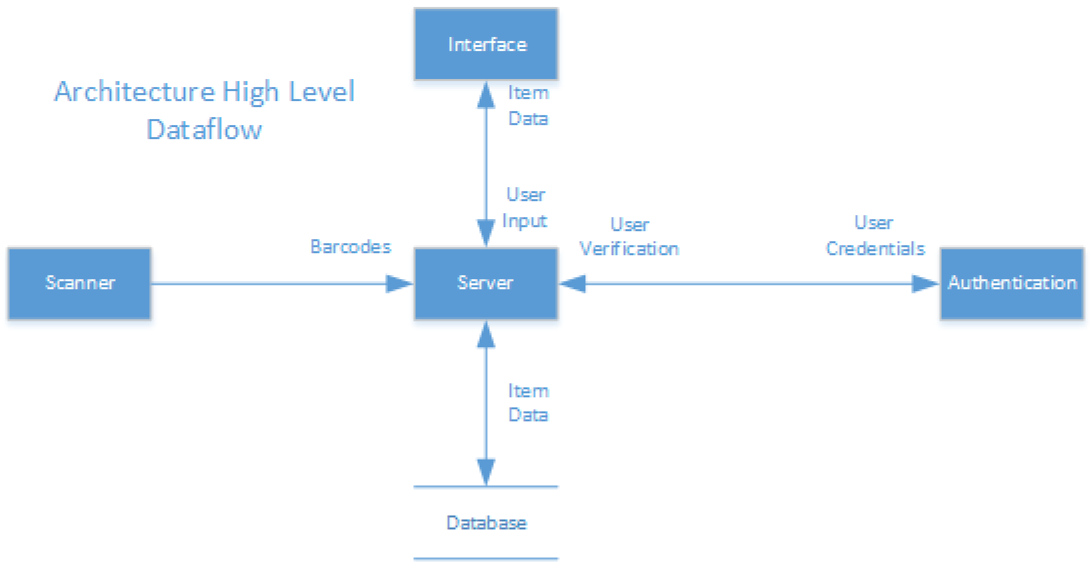


Figure 1: High-Level Dataflow of the System Architecture

Detailed Description of Components

Interface

Web Application

<u>*Design Entity Name*</u>: Web Interface
<u>*Type*</u>: Interface
<u>*Purpose*</u>:
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.

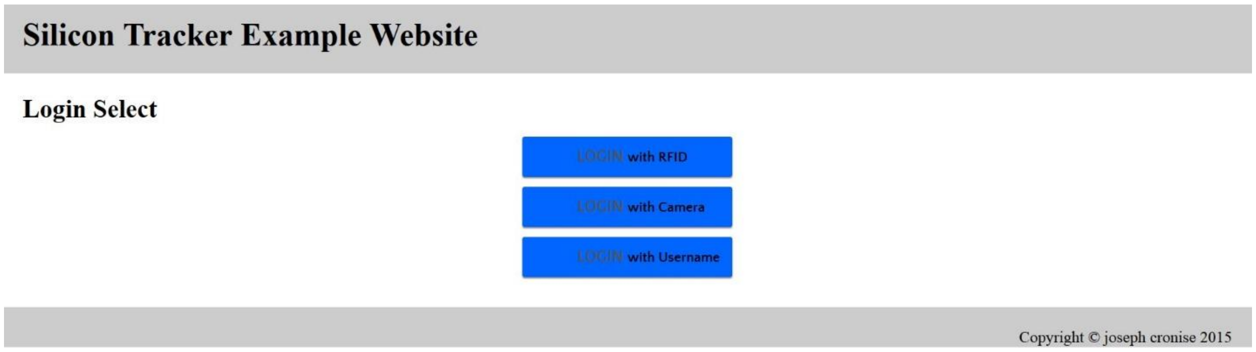


Figure 2: Users will be presented with the option to choose their login method from this page.

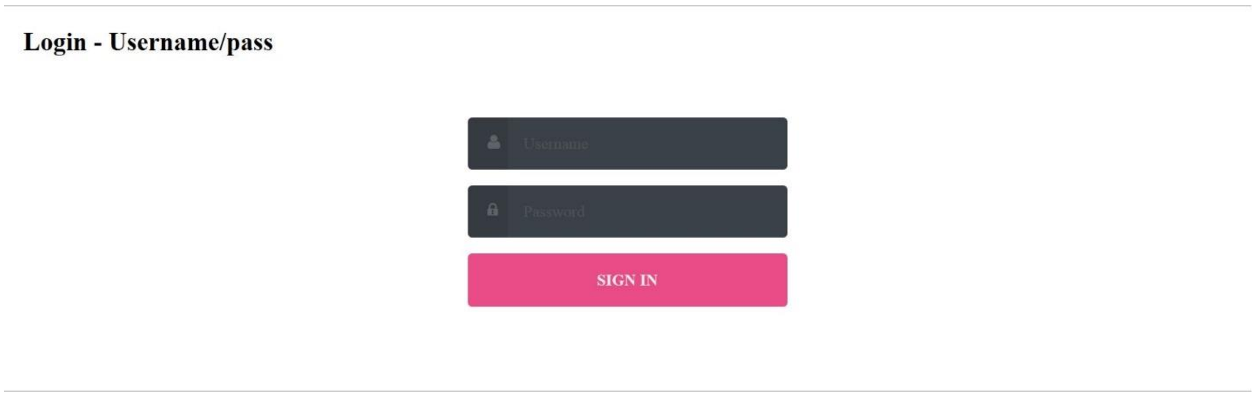
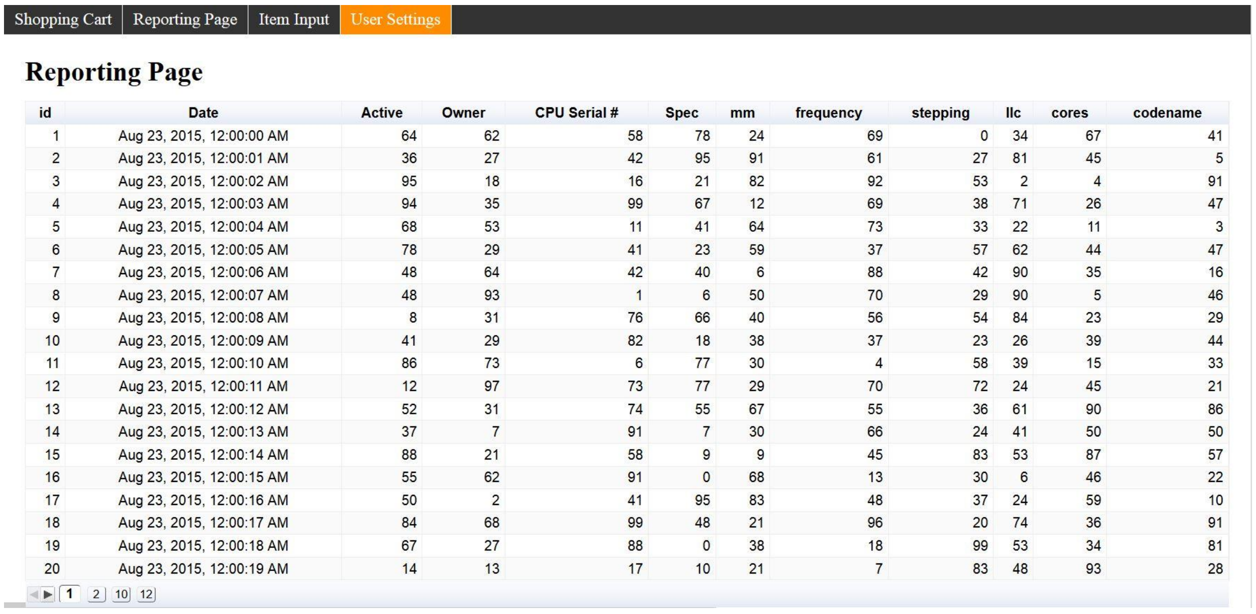


Figure 3: If users choose to login using their username/password they will be presented with this style of page.



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.

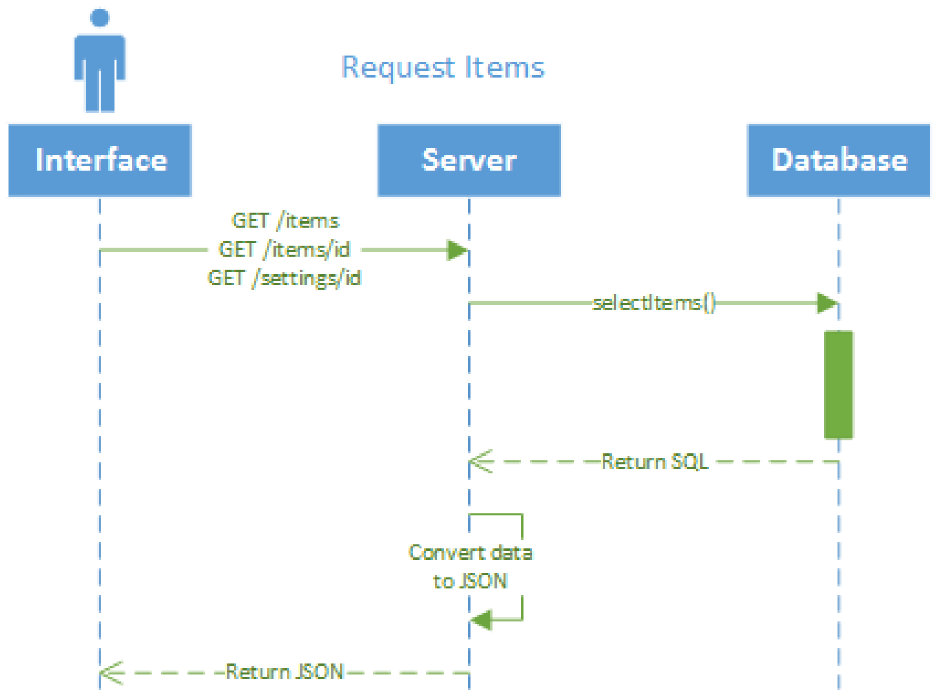


Figure 5: REST Sequence Diagram

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.

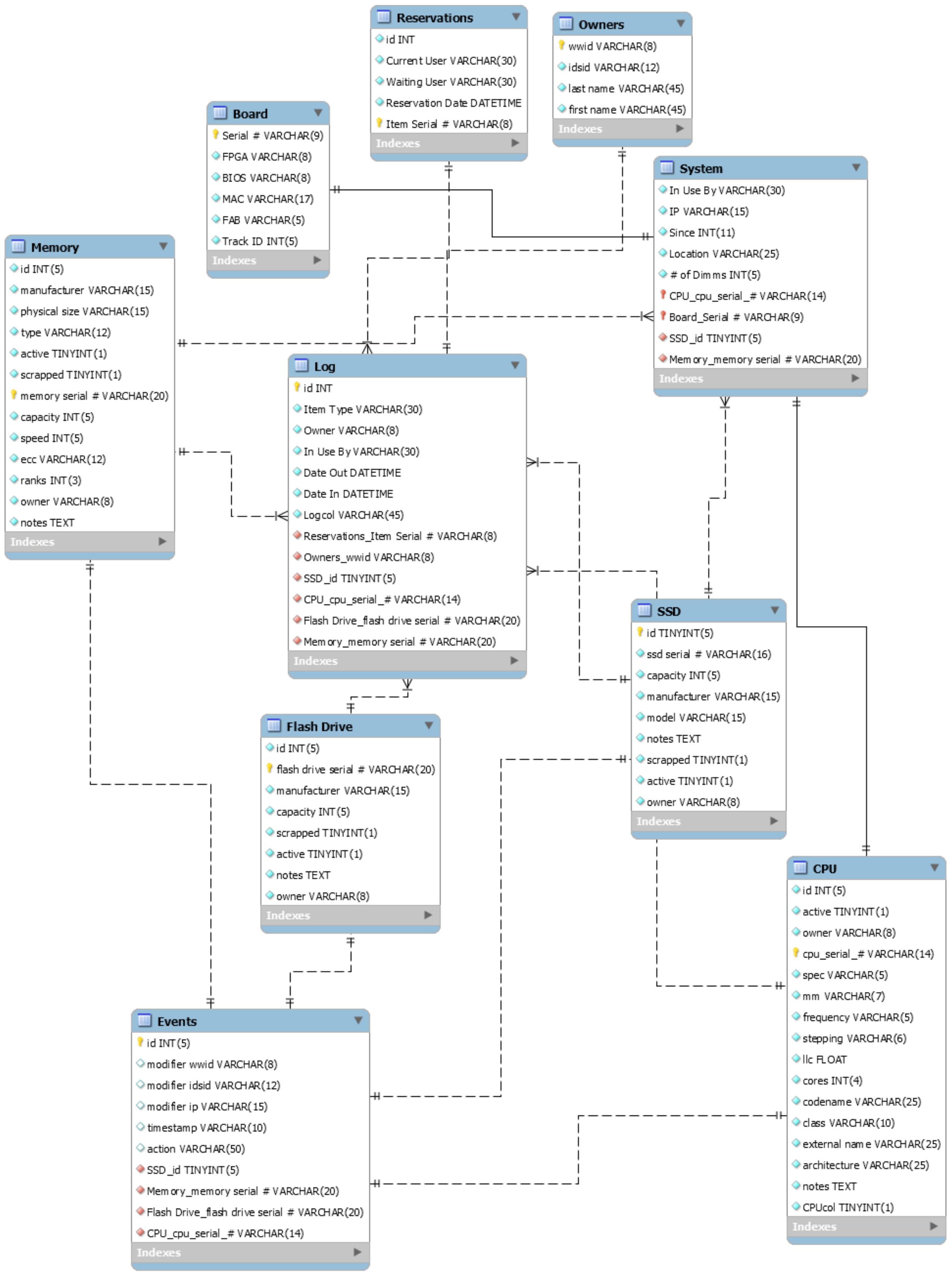


Figure 6: ER diagram detailing the specifics of the Inventory Database

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.

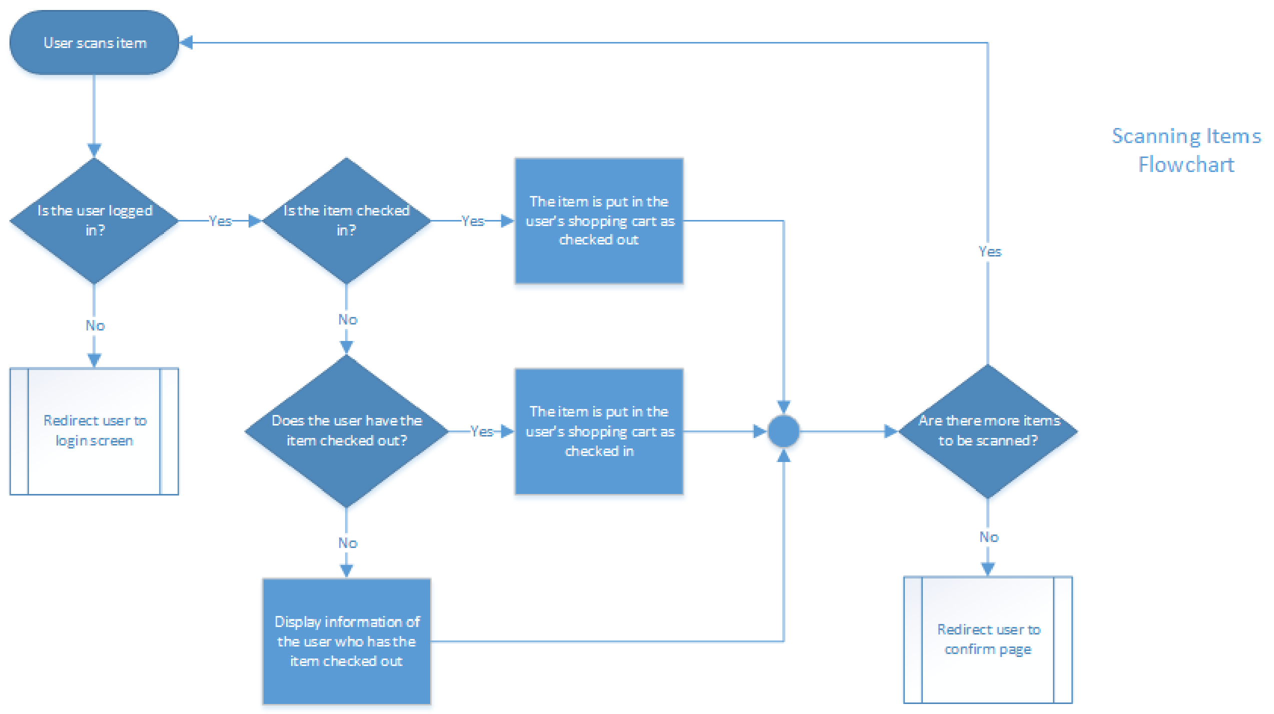

User scans item
Is the user logged in?
Yes
No
Redirect user to login screen
Is the item checked in?
Yes
No
The item is put in the user's shopping cart as checked out
Does the user have the item checked out?
Yes
No
The item is put in the user's shopping cart as checked in
Display information of the user who has the item checked out
Are there more items to be scanned?
Yes
No
Redirect user to confirm page
Scanning Items Flowchart

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.