Senior Design Final Report

Danfoss Forklift Request Application

May14-34

Date: April 27th, 2014

Customer: Danfoss Power Solutions - Manufacturing IT
Brad Rosenhamer - Manufacturing IT
brosenhamer@danfoss.com

Advisor: Samik Basu
sbasu@iastate.edu

Project Team: Jonathan Carlz
Ryan Sanders
Tyler Jensen
Sean McCullough

Table of Contents

Project Abstract

Danfoss Power Solutions is an engineering and manufacturing firm that provides OEMs (original equipment manufacturers), such as John Deere and Caterpillar, with hydraulic components for their final products. Danfoss requires a method to facilitate forklift dispatch at their Ames facility.

Previously attempted solutions included two-way radios and cell phones, however, both were ineffective. The radios and phones were difficult to hear in the factory and presented safety concerns.

The project described in this document involved developing a web-based request and dispatch solution, deployable over a local network. Assembly line employees enter requests at PCs near their workstation, which forklift operators can respond to on tablets mounted to the forklifts.

Key Definitions

Administrator
A Danfoss employee who has supervising authority over the application. He has access to the application through a browser on a PC.

Application
The software solution designed and created by the May14-34 group to solve Danfoss' forklift dispatch problem.

Assembly line employee
A Danfoss employee who works on an assembly line. These employees access the application through a browser on a PC.

Backend
A portion of the application that interacts with requests and request states directly.

Client
A web browser used to access the application. Examples include Microsoft's Internet Explorer and Apple's Safari.

Deployment
An instance of the application running in a production environment.

Forklift operator
A Danfoss employee who operates a forklift, and accesses the application via a tablet mounted on the forklift.

Frontend
A portion of the application that controls client access to the Backend portion of the server

Request
Also referred to as a forklift request. A request is what an assembly line employee creates and a forklift operator processes.

Request state
A request can be in one of seven request states: opened, assigned, in-progress, completed, canceled, abandoned, and rejected. These states describe where a request is in its lifecycle.

Server
The server is the computer that accepts and responds to client calls.

System Requirements

The FRA has a variety of functional requirements. These requirements come from three stakeholders: Danfoss and the Senior Design Syllabus. The group has two subsets of their requirements: functional and nonfunctional. Functional requirements include "physical" (with respect to software) abilities the application must provide. Non-functional requirements include attributes such as reliability, responsiveness, and accessibility.

Functional

Functionally, the FRA must allow assembly line employees to create Forklift Requests and forklift operators to respond to those requests. Additionally, administrators must be able to perform operations such as viewing forklift request log data in both tabular and graphical form.

Finally, the application must be fully compatible with Internet Explorer 8 and Safari Mobile web browsers. These browser requirements greatly impacted design decisions and chosen technologies.

Non-Functional

Non-functionally, the Forklift Request Application must be reliable, responsive, and accessible. To ensure reliability, the team created coherent unit tests covering the code base. These unit tests verified both normal and illegal input to ensure end users receive the best experience possible. To ensure responsiveness and accessibility, data is stored efficiently in an indexed form, and populates the user interface in an asynchronous way, to ensure only non-redundant data transfer and processing. Finally to ensure accessibility, icons were created to help color blind users of the application.

Project Design

Requests

When an assembly line employee needs assistance from a forklift, a request is created using the Forklift Request Application (FRA). Requests states are used to track the lifecycle of a request. A request can make legal transition from one state to another at any time; however a request only exist in one state at a time. The requests states are described in detail below.

Request States:

Each state has an associated color. Symbol are also provided to aid color-blind users.

Name	Color	Symbol	Note
OPENED			Request has been created, but not yet accepted by a forklift operator
ASSIGNED			A forklift operator has accepted the request
ABANDONED			Request has been in forklift operator's queue for more than 10 minutes. It is marked as abandoned and placed back in the main queue
IN_PROGRESS			Forklift operator has begun working on a request
COMPLETED			The task has been completed
REJECTED			The request information was incomplete and the request could not be completed
CANCELED			The request is no longer needed

Request State Flow

The State Flow Diagram, shown in figure 1, shows the request states, valid transitions, and actions initiating these transitions. When an assembly line user creates a request, it begins in the **opened** state. Then, a forklift operator may select the open request and **assign** it to his/her work queue. The forklift operator can pick a request from his/her queue to complete, causing the request to enter the **in progress** state. If the forklift operator fails to complete the request in 10 minutes, the request is **abandoned**, and returned to the main queue. Forklift operator may also **reject** a request if it contains incomplete or incorrect information. Likewise, assembly line users may **cancel** a request that is no longer needed. When a request is completed by a forklift operator, it is marked as **complete.**

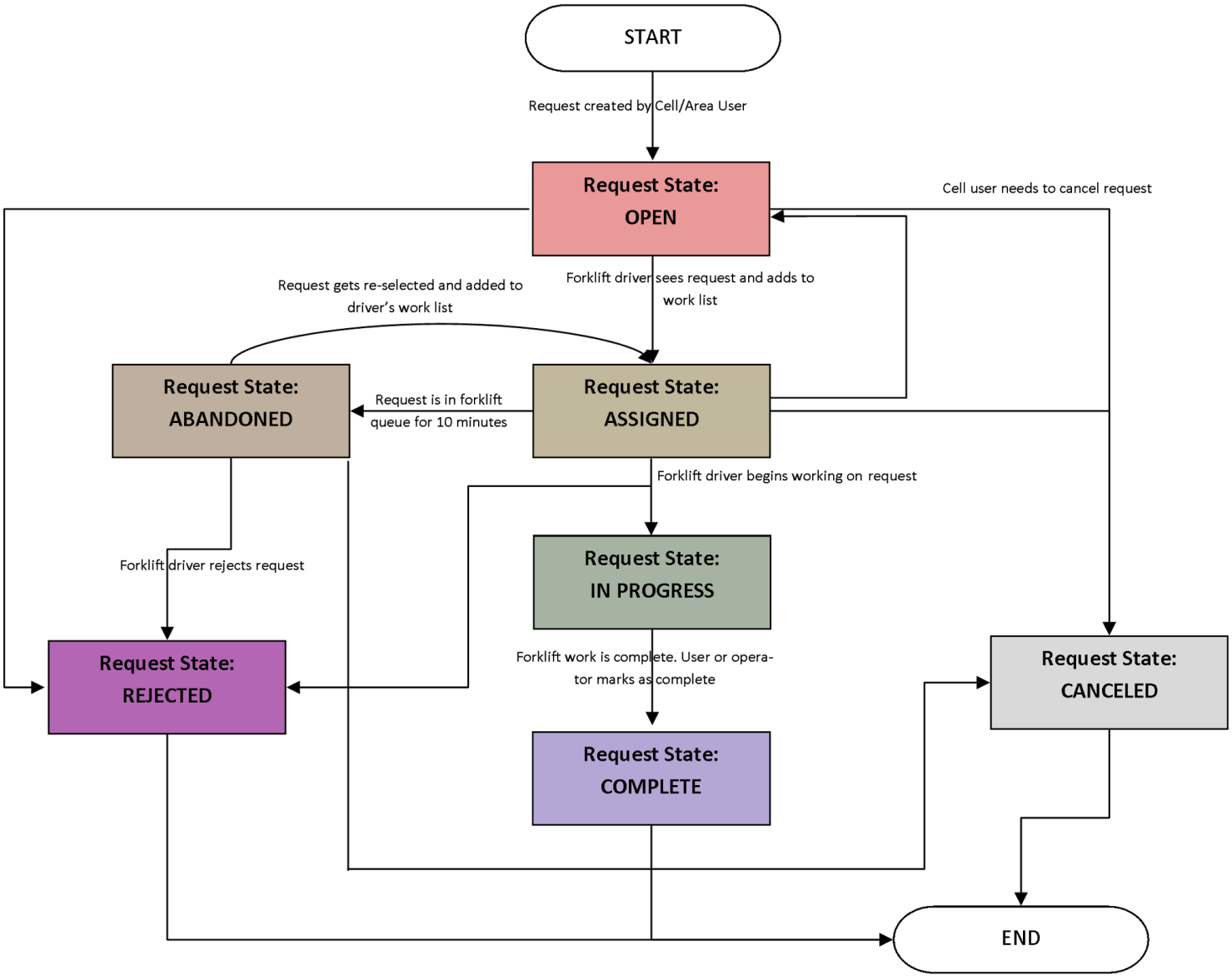


Figure 1 - Request State Flow Diagram

Enterprise Architecture Component Diagram

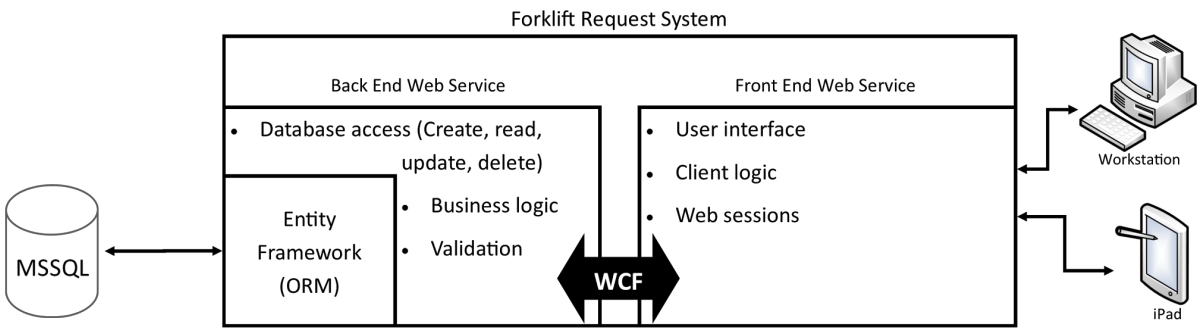


Figure 2 - Enterprise Architecture Component Diagram

Beginning on the left side of the diagram, the first component of the system is a Microsoft SQL database. The database was created in Microsoft SQL Server 2012, which conforms to Danfoss' requirements and current standards. The schema for the database is shown in figure 2 and further explanation can be found in the Database Schema section.

Entity Framework is a component of the Backend Web Service, which performs several major functions. This component provides the Frontend Web Service an interface to the database. This database interface is a security feature, as it prevents direct access to application data from the user facing side of the application. It also allows for validation of data, and the logic facilitating the flow of requests in the application.

The Frontend Web Service is the component that handles web sessions, which are used for user authentication. The front end provides the user interface, including how the information is prepared and presented.

The communication between the Backend Web Service and the Frontend Web Service is done through Windows Communication Foundation. WCF is a Microsoft tool that facilitates API calls between services running in a distributed system. In the case of the FRA, WCF allows the Front End Web Service to invoke methods of the Back End Web Service, even if they are running on separate servers.

The final component of the system is the client, which runs in the web browser on PCs or tablets. The PCs are located near each assembly line, and allow access to the assembly line view. The tablets are mounted on the forklifts, and allow operators to access the forklift view of the application. The tablets selected for the project were iPads, as they are the standard tablet device for Danfoss. However, an iPad is not strictly necessary, because the system is a web application, and can be accessed from any modern browser.

Database Schema

There are two main sections of the database schema: one for requests, and the other for users. Starting in the lower left of figure 3, the Role table contains roles a user can have - either as a forklift operator or assembly line employee. The user table contains the ID number and username for all users in the system. The Role and User tables are linked by the UserRole table, which associates UserID and RoleID for each user.

There are five tables in the Request section. Beginning in the upper left, the RequestStates table consists of the valid states a request can have. Although this information is not strictly necessary to exist in the database, it allows for data validation through table relationships. Next, the different assembly lines are stored in the Area table, and individual PCs are associated with assembly lines in the AreaDeviceMapping table. Actual request objects are stored in the Request table, and each request state change is recorded in the RequestStatus table. This means that a request can have multiple entries in the RequestStatus table, one for each state change.

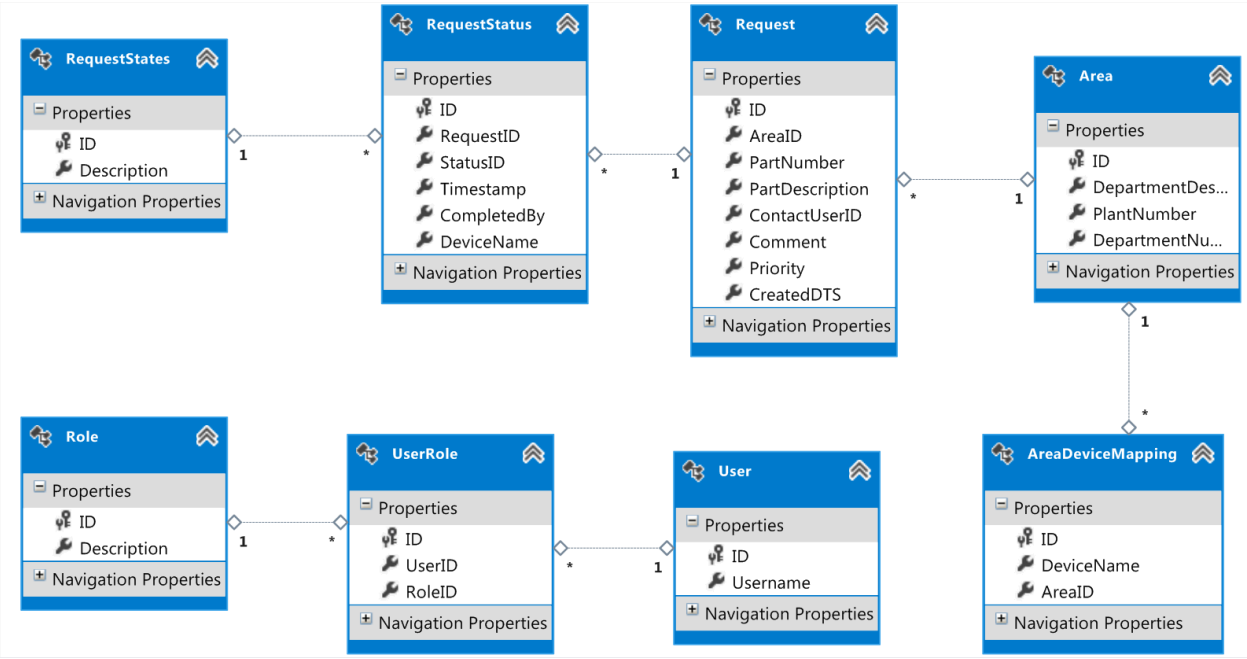
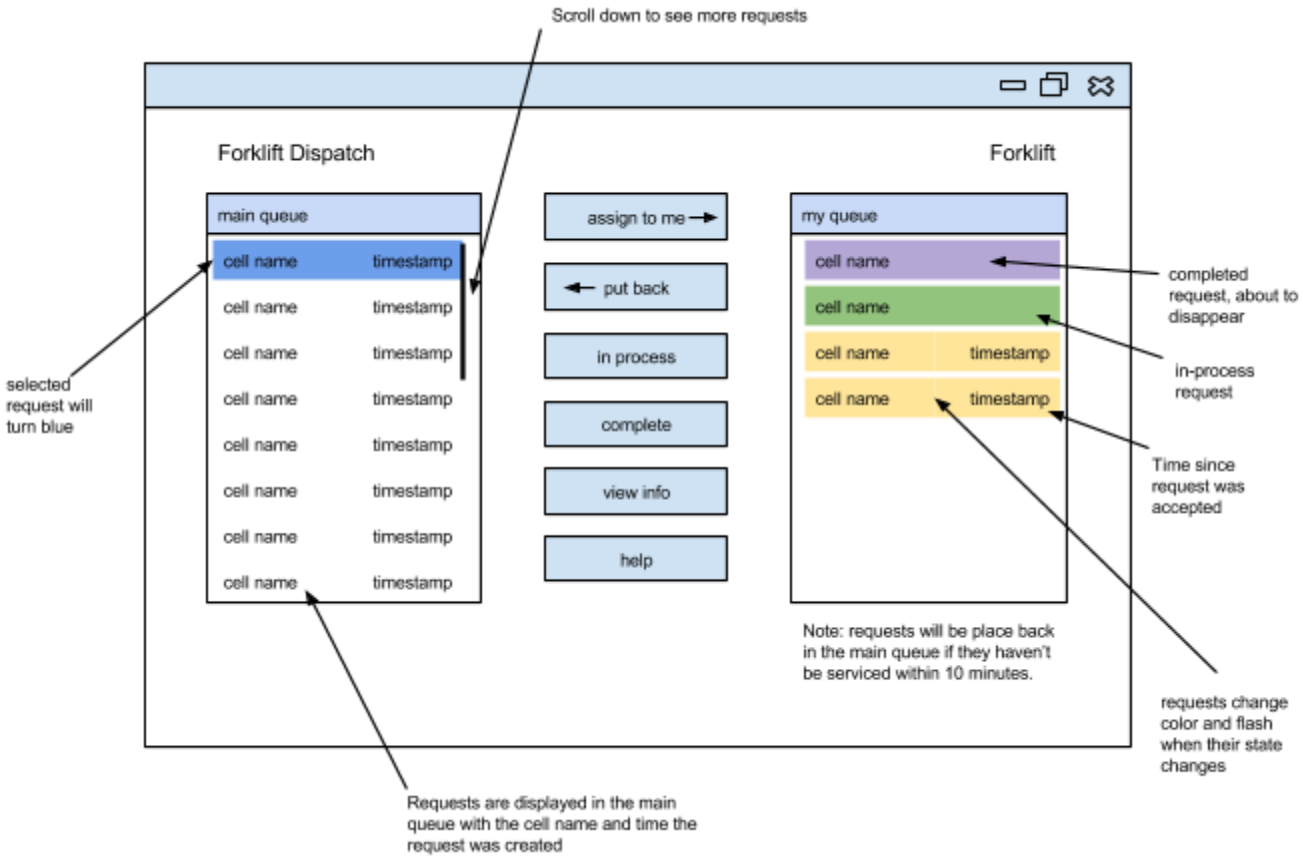


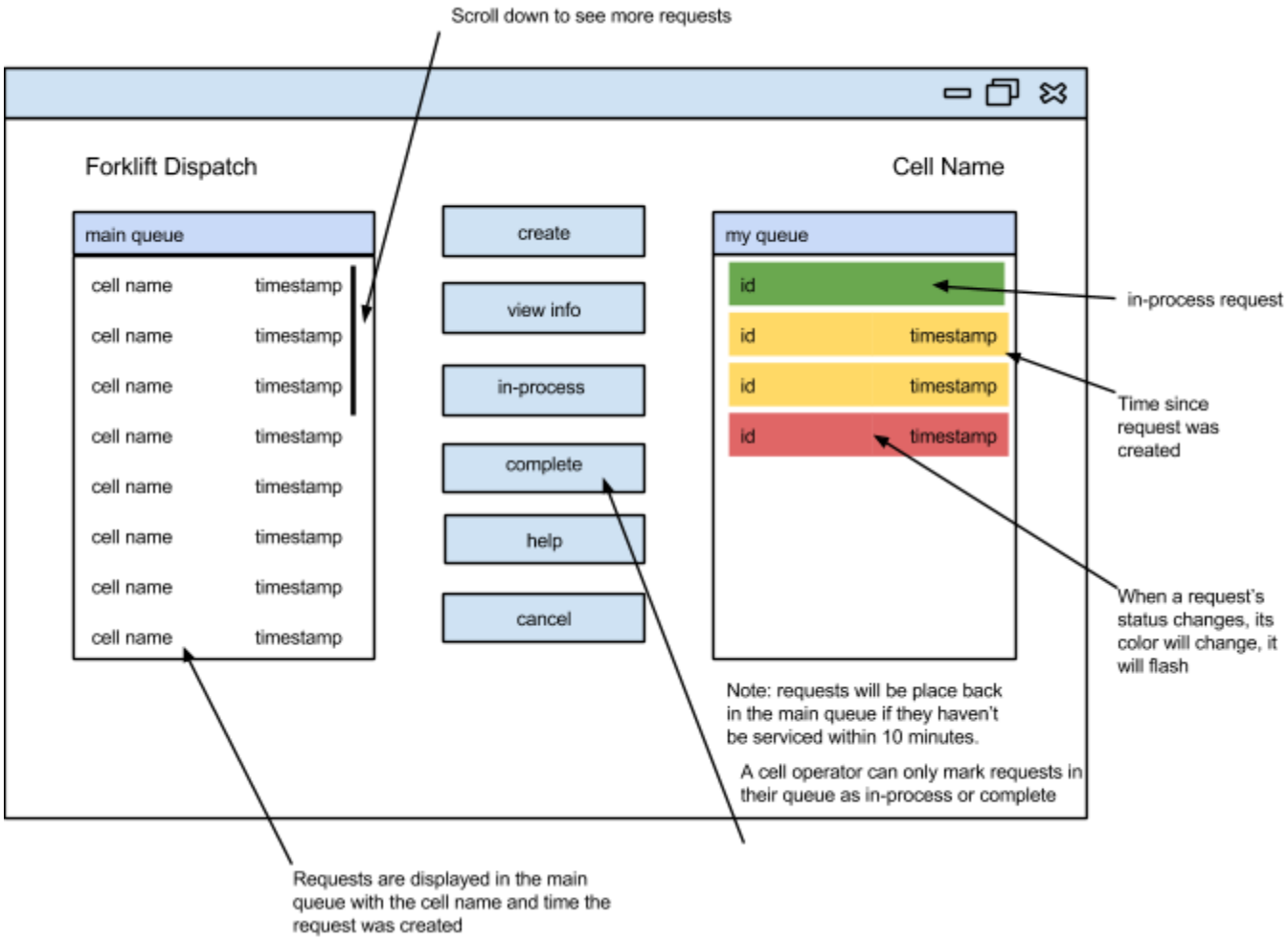
Figure 3 - Database Schema for the Forklift Request Application

Screen Sketches

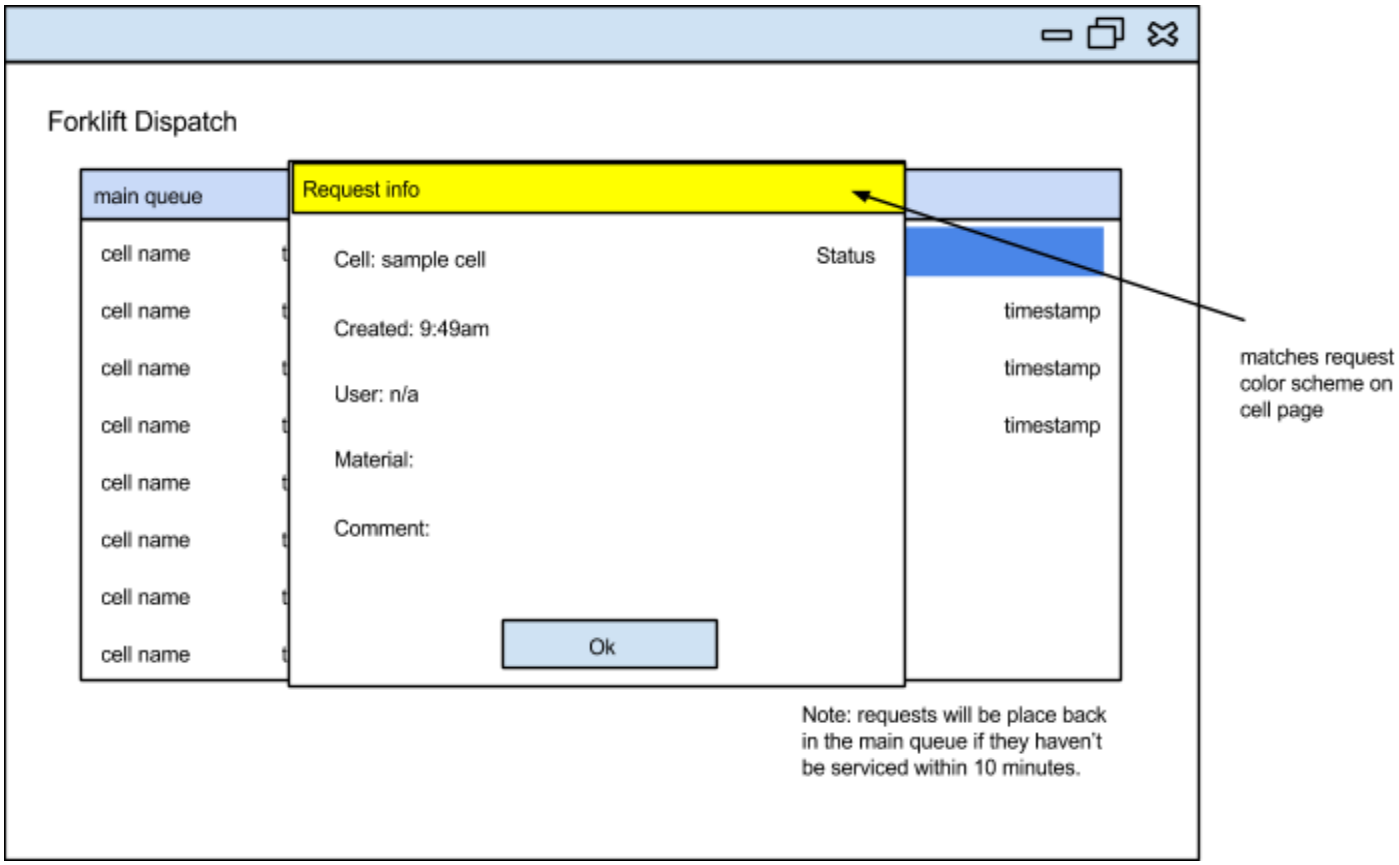
Forklift Operator View



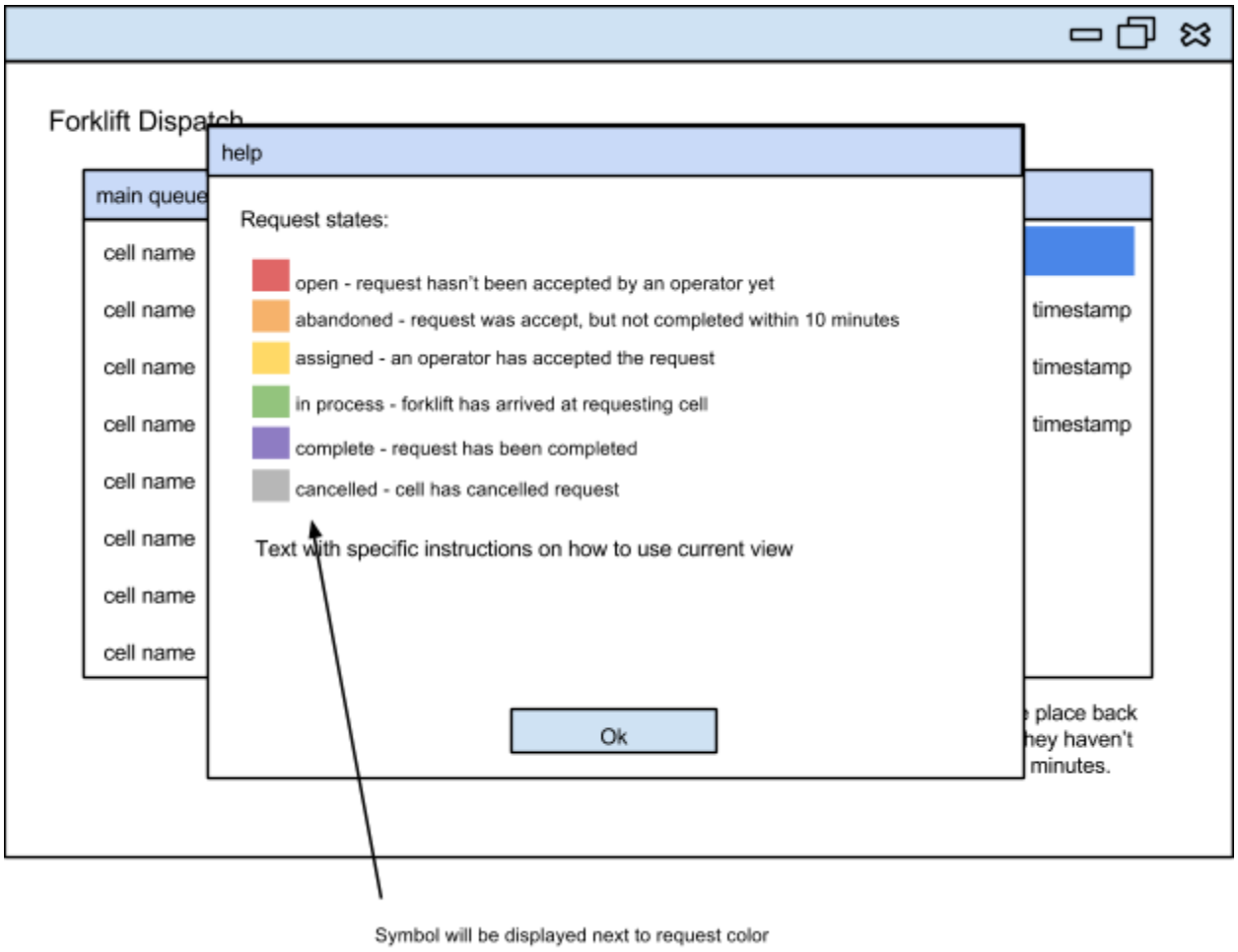
Assembly Line Employee View



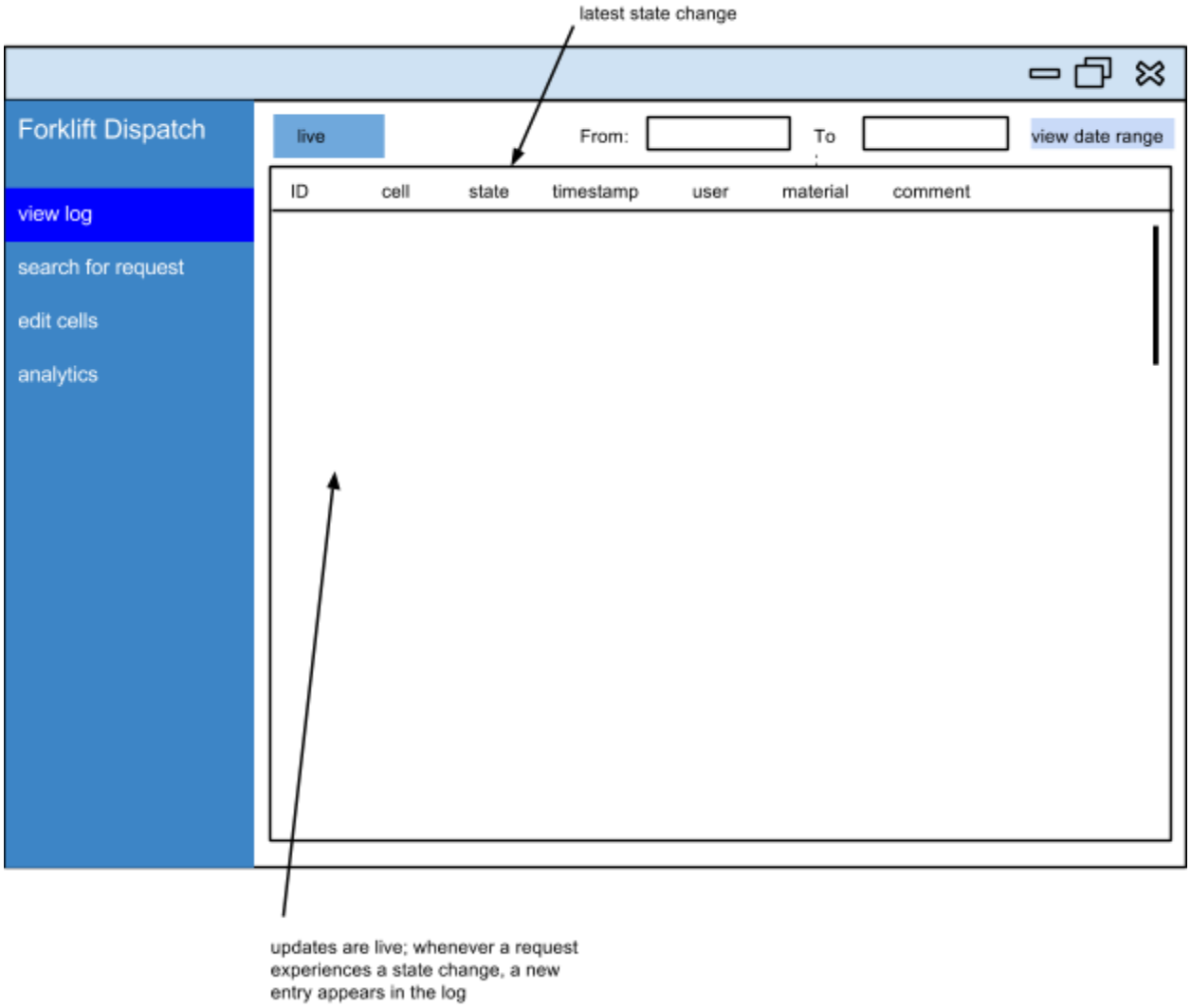
Request Detailed Info Screen



Help Screen


Forklift Dispatch
help
main queue
cell name
cell name
cell name
cell name
cell name
cell name
cell name
cell name
Request states:
open - request hasn't been accepted by an operator yet
abandoned - request was accept, but not completed within 10 minutes
assigned - an operator has accepted the request
in process - forklift has arrived at requesting cell
complete - request has been completed
cancelled - cell has cancelled request
Text with specific instructions on how to use current view
Ok
timestamp
timestamp
timestamp
place back
hey haven't
minutes.
Symbol will be displayed next to request color

Administration Screen



Standards

Danfoss Standards

Danfoss imposed many standards over the project. They included coding style, comment formatting, color choices, and logo placement. First, Danfoss has a strict coding/commenting style enforced by a software application called StyleCop. This requirement aids code maintenance, developer productivity, and automatic documentation. The second standard specified color choice and logo placement. This standard creates a uniform look and feel across internal Danfoss applications.

W3C Standards

Web development standards were strictly adhered to, as the FRA is a web-based application. The World Wide Web Consortium (W3C) is the main international standards organization for the web. Specific standards they maintain are web design and applications, web architecture, XML markup, and web enabled devices. The most relevant standards to the project are HTML, CSS, and AJAX.

Software Development and Evolution Standards

The project utilized a waterfall methodology to structure the development process. First, the team met with the client several times to hash out the functional and nonfunctional requirements of the application. Next, several designs for each component were brainstormed and carefully considered. Based on the requirements of the system and the given standards for the application, decisions were made regarding the final design of the system. The design was taken to the client for approval, and after minor revisions, the design was approved. The design included: the database schema, the interface for both the Frontend and Backend web services, user interface sketches, and a test suite framework.

Next came the implementation component of the project. Due to the planning in the design phase, different components of the application were able to be developed in parallel. For example, the software interface that was defined during the design phase allowed the Backend component and the Frontend component to be written at the same time. As features of each were implemented, they were combined and tested to verify basic functionality. Once the basic functionality of the system was completed, the unit tests and user interface tests were run. The results of these tests determined whether the project met the requirements specified by the client. After any modification to the application, whether logic or user interface related, the suite of tests was run to ensure the system still met the basic requirements.

The last step of the waterfall methodology is to maintain the application. After implementation at Danfoss, there were small tweaks made to the application. These were necessary because of the minor differences between the development and production environments.

Implementation Details

Views

Three distinct user views are implemented in this application. They are the “Forklift Operator View”, the “Assembly Line Employee View”, and the “Administrator View”. The first two are based on a similar layout, while the Administrator View is unique.

Forklift Operator and Assembly Line Views

As previously stated, the Forklift Operator and the Assembly Line View share a layout. These two views are displayed using the Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS). HTML and CSS are standard in web development (see the standards section on the previous page). To receive the HTML and CSS, a forklift operator or assembly line employee navigates to the server hosting the application. The server then sends the HTML and CSS to the user’s browser. The browser automatically renders both into content similar to the images in the Screen Sketches section of this document.

HTML and CSS are static, so JavaScript is used to make dynamic changes to the View. JavaScript is heavily intertwined with HTML and CSS and allows dynamic modifications the page. In addition, JavaScript can communicate with the server post page load using a JavaScript XMLHTTPRequest object, commonly referred to as an AJAX request.

This application makes heavy use of JavaScript. All changes a user sees post page load are a result of JavaScript code executing and modifying the existing HTML and CSS. Using this method reduces screen flickering, decreases network traffic, and aids application scaling.

As previously mentioned, when a view needs to exchange data with the server, it uses an AJAX request. These AJAX requests go directly to the Request Controller on the application server. The Request Controller processes the request and returns an appropriate response. This response is in JSON, which stands for JavaScript Object Notation. JSON is used to represent JavaScript objects in a serialized text form. Upon receiving this JSON response, the view parses it into a JavaScript object, and performs any necessary operations.

Administrator View

In contrast to the previous views, the Administrator View follows more of a traditional web standard. The Administrator View still uses HTML and CSS, but the user clicks hyperlinks to modify the view. After a user clicks a hyperlink, the server responds with new HTML and CSS, and the page is reloaded and rendered.

Controllers

The controllers of the application are what glue the Views and the Model together. View actions requiring changes in the server state are forwarded to the Model, which executes these changes. The application makes use of three separate controllers: Authentication, Request, and Expired Request.

Authentication Controller and Request Controller

The Authentication and Request Controllers communicate with the Views using the HTTP protocol. They each communicate with the Model using Microsoft Windows Communication Foundation.

The Authentication Controller wraps around the Request Controller, ensuring only authorized access to the model. Upon any unauthorized attempt, this controller immediately redirects the user to the login page.

The Request Controller facilitates all forklift requests actions; this is the heart of the application. There are three actions: create request, update request state, and retrieve request events.

Creating a Request

As mentioned in the Views section, when a user wishes to create a new request, the user enters the appropriate parameters into the view. The view forwards the request to the Request Controller via an HTTP call. The Request Controller parses this HTTP call, and extracts the needed parameters to create the request. The parameters are validated and passed to the model, where the request is created. Upon successful creation of the request, the model returns the request's unique ID. The ID is serialized into JSON, and returned to the view.

Updating a Request State

As mentioned in the Views section, when a user wishes to update the state of a request, he selects said request and click the desired transition button. The view constructs an HTTP call with the needed parameters (request id and transaction code), and sends it to the Request Controller. The Request Controller parses the HTTP call, and extracts the parameters needed to update the state of the request. The parameters are validated and passed to the model, where the state of the request is updated. Upon a successful update of the state of a request, the controller creates a JSON object. This JSON object is wrapped as part of a HTTP/JavaScript response and sent back to the view.

Retrieve request events

A view can ask for new request events in the system. Each view does this automatically in a loop, so that they have the most up-to-date state of the Model. To retrieve all new events, the view creates a HTTP call to be sent to the controller. Upon receiving the call, the controller asks the model for all requests. If there are no new events, the controller will continue to poll the model for updates. This is referred to as Long Polling, and is described in detail in Appendix II. The resulting serialized JSON array, which is wrapped as part of a HTTP/JavaScript response,

is sent back to the view. It should be noted that this action takes an optional Timestamp called "sinceTime". The "sinceTime" parameter can be thought of in the following way: "Give me all requests changes since time X". This allows a view to receive only the updated part of the dataset. This is useful and preferred if the caller already knows about all events up to time X.

Expired Request Controller

The Request Controller and views do not have the ability to delete requests or set them to the abandoned state. This is where the Expired Request Controller comes into play. The Expired Request Controller runs silently on the server, and does not need authentication to access the model. There is no view associated with this controller.

The purpose of the Expired Request Controller is twofold: first, the expired Request Controller checks all requests in the assigned state to see if any have been assigned for longer than 10 minutes. If so, the controller signals the model to set those requests as abandoned.

The second task of this controller is to remove all requests that are in any of the three final states: completed, canceled, or rejected. Before removing a request, the service first extracts all data about a specific request and writes it to the log table. There are two reasons for this: first, logging the data allows for detailed analytics to be collected. Second, requests in their final states are no longer needed for general use of the application, and removing them increases performance. Both components of the Expired Request Controller are best effort services, meaning they do not wait for a response from the model.

Model

The Model is the innermost layer of the application. As mentioned previously, the controllers are written in C# and manipulate the data in the model. C# is an object oriented programming language, meaning the forklift requests are treated as discrete "objects" that represent real world items. For example, the FRA has request objects that contain attributes including part number, description of a part, person requesting the part, and assembly line where the part should be delivered. If the application stops, all the information contained in these objects is lost, unless it is stored to a physical medium, such as a hard drive. To ease writing and retrieval of the information stored on the hard drive, a database is used.

Per Danfoss standard, information is stored in a Microsoft SQL database, a relational database. Instead of being stored as objects, like in the application, a relational database stores information in logical columns and rows. Columns contain information of the same type, and are labeled with a column name, such as part number or requesting user. Rows span across all columns and represent a complete artifact. In the FRA, a request object contains the user requesting the part, part number, part description, where the part should be delivered, an optional comment, and a timestamp.

To map the C# objects to the Database an Object Relation Map (ORM) called Entity Framework handles the mapping between the database, or model, and the object used in the C# application.

Testing

There were five types of testing used during the development of the FRA: Frontend, Backend, Scalability, Connectivity, and Beta.

Front End

Frontend testing involves testing the user interface (UI). This type of testing includes both valid and invalid use cases. These tests are conducted by simulating user input, and checking for the desired UI behavior. For example, “the color of Item X after Button Y is clicked should be blue”. Inputs include, but are not limited to: mouse clicks, finger taps (tablet), and keyboard input. UI testing is resource intensive, and error prone if performed manually. Furthermore, the number of individual UI tests grows exponentially with the number of possible UI actions. Because of this, a tool called Selenium was used to automate this process. Selenium records a given UI action, as well as the state of the UI after the action is performed (e.g. move cursor to Button X, click it, and ensure Button X is deactivated). Using Selenium, a collection of tests can be compiled and run with the click of a button.

The power of automated UI tests become apparent when the application is expanded or slightly changed. Running the tests after a change confirms that previous functionality has not been affected.

To date, all Frontend tests created pass with the current version of the source code.

Back End

Frontend tests are only able to test UI content presented to the end user. To test the Backend component of the project, the team used the nUnit framework. nUnit is a unit testing platform for the .NET framework. Unit tests are used to ensure specific functions behave correctly, by calling a function in the application, and comparing the actual output against the expected output. For example, when testing a function `squareRoot(int)`, a nUnit test might be `squareRoot(16) = 4`. nUnit looks for that function to return `4`, and will throw an error if this is not the case.

In this project, Backend testing involved request state transitions. For example, a completed request should not go to the in-progress state. On the other hand, an in-progress request should always be able to transition to the completed state.

As stated above, manually conducting these tests would be tedious and error prone. A request can be in one of 7 states, meaning there are 49 (7 * 7) possible transitions, 37 of them illegal, all of which had to be tested.

Scalability

Frontend and Backend tests only answer the question “Does the application function as expected?” “How well does it work?” is an equally important question. This is where scalability and connectivity testing come into play. Scalability tests ask the question “How well does the application work when X users are using the application concurrently?” Given the testing hardware, an acceptable network connection, and software algorithms used throughout the application, it was determined around 30 users could concurrently use the application without noticeable lag (greater than a few hundred milliseconds).

Connectivity

Furthermore, how the application responds to network inactivity was also tested. Basically, whenever a client computer has trouble reaching the server, it displays a message to the user, disables input, and continues trying to contact the server until a connection is made. At that point, input is re-enabled and the error message is removed. Connectivity testing is vital to this application as half its users will be using the application over WiFi and will be moving between wireless access points in the facility.

Beta

Last but not least is Beta Testing. Beta testing typically consists of giving early versions of the application to actual users. This is done to solicit feedback, and help identify an bugs that slipped through the other testing phases. In beta testing this project, an early version of the application was shown to Danfoss employees, and used their feedback to improve the user experience.

Appendix I: Operation Manual

Requirements:

- Microsoft SQL Server 2012
- Microsoft IIS 7
- Administrative privileges to install applications

Installation

1. Open Microsoft SQL Server Management Studio and connect to the MSSQL server where the FRA database will be hosted.

2. Create a new database called “Forklift Request App”

3. Right click the newly created database and click “Import”

4. Select the FRA.sql file from the install folder and click ok. SQL server will build the database schema.

5. Open the Microsoft IIS manager console.

6. Right click the “Web Sites” item on the left and click “Create new web site”

7. Enter “Forklift Request Application” as the site name

8. Enter the desired URL and port for the system. Click OK.

9. Return to the install folder. Double click the Backend installer.

10. Follow the prompts of the Backend Installer, which will place the appropriate files in the web publishing folder created by IIS in step 8.

11. Once the Backend installer finishes, double click the frontend installer.

12. Follow the prompts of the Frontend installer

Starting the Server

IIS normally automatically starts newly created websites. In the off chance it does not, or your wish to stop/restart the server go, to Microsoft IIS and complete the following steps:

1. Right Click on the Forklift Request Application web site
2. Select “Stop” or “Restart”

Requests States

To use the Forklift Request Application, it is important to understand the states requests can be in. Each state has an associated color and symbol. For more explanation about request colors and symbols, see the help screen, available in both the forklift operator and assembly line views by clicking or tapping the “help” button.

- **Open** - Requests are initially created by assembly line employees. After a request is created, it is in the open state. Open requests are placed in the main queue, and are viewable to all forklift operators. The assembly line employee who created the request is referred to as the request owner. All requests created by an assembly line employee can be viewed in the user’s personal queue.

- **Rejected** - If a request is incomplete or contains incorrect information, a forklift user can mark it is rejected. This change is reflected on the request owner’s personal queue, along with an explanation for why the request was rejected. The request owner can then create a new request with the correct information.

- **Cancelled** - If a request owner no longer needs a request to be complete, he can cancel the request. It is removed from the request owner’s queue, and if the request is open, it is removed from the main queue. If the request has been assigned, it is removed from that forklift operator’s personal queue.

- **Assigned** - A forklift operator can select an open request, and add it to his or her personal queue. When a request is “assigned”, the change will be shown on the main queue for all forklift users, and in the personal queue of the request owner.

- **In-progress** - Once a forklift operator starts work on a request, the forklift operator can mark the request as “in-progress”.

- **Abandoned** - If a forklift operator is unable to service a request in 10 minutes, the request becomes “abandoned”. It is removed from that forklift operator’s personal queue, and placed back in the main queue, so another forklift operator can complete the request.

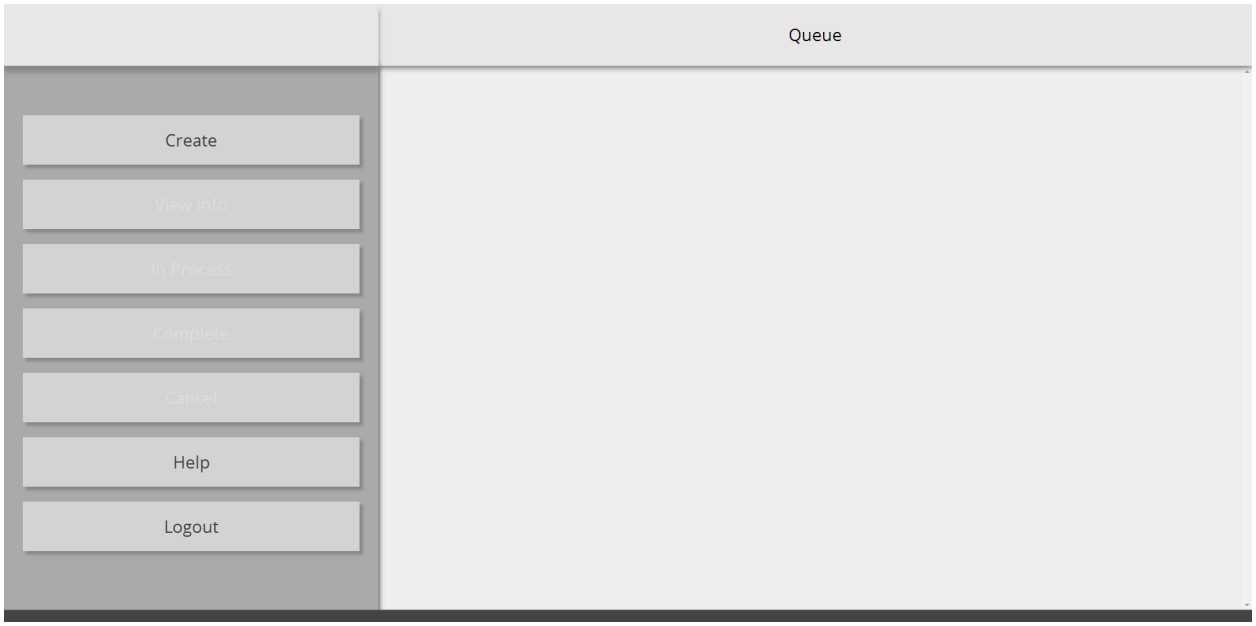
- **Complete** - once a request has been completed, it can be marked as complete by the request owner or forklift operator

Assembly Line User

Login

1. Navigate to http://may14-34.ece.iastate.edu/Area.aspx

2. If you see a screen similar to the one below, you have successfully logged in. The buttons on the left show your available actions.



If the application prompts for a login, your system is not configured correctly. Please contact the Danfoss IT Help Desk at x1234. Provide the following information:

- Your Name
- An extension where you can be reached
- The name of the PC you are on. This will be of the form PCxxxxx
- The name of the department and/or department number

IT will configure the given PC to access the Forklift Request System. Once configured, you will be able to use the system.

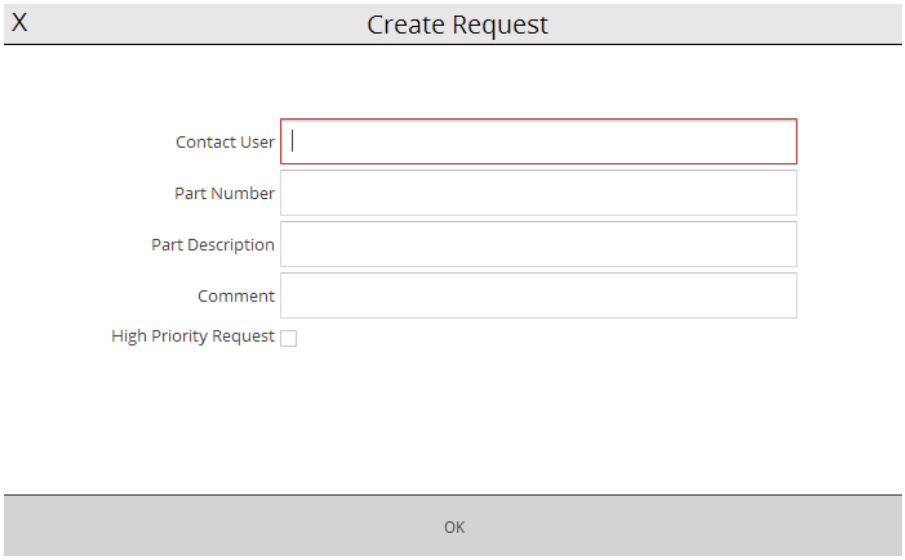
Understanding Assembly Line User Commands

Once successfully logged in, there are several commands, some of which may not be available based on the states of the requests in the queue.

Create a request

1. Click the “Create” button

2. The “Create Request” dialog will appear.



3. Enter the following information:

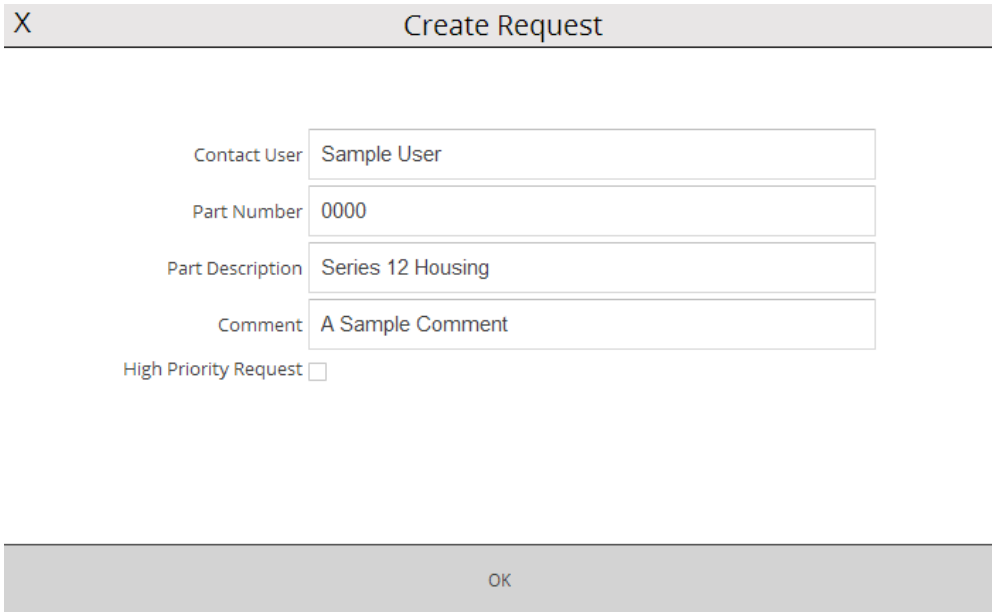
   Contact User - The person to contact if there is a problem with this request. This will likely be your name.

   Part Number - Enter the full part number of the item you are requesting. Requests with incomplete part numbers will be rejected. If you are requesting an “in area” operation, enter 0000 for this field.

   Part Description - Enter the description of the part. Be specific, as this will help the forklift operator ensure prompt retrieval of the correct material. For example, use “Series 12 Housing” instead of “Housing.”

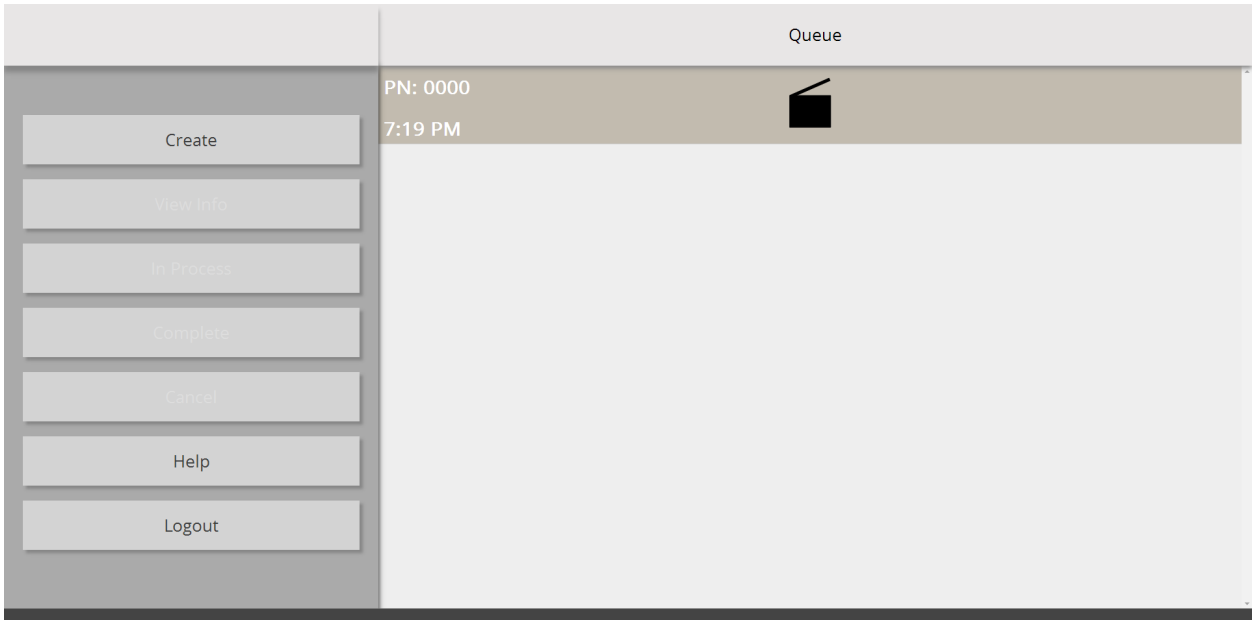
   Comment - This is optional. If you have any special instructions for the forklift operator, type them here. Limit: 450 characters.

   High Priority - check this box if this is a high priority request. Typically, a request should only be high priority if production will stop if the request is not serviced within 15 minutes.



4. Click the “OK” button at the bottom of the screen. If there is a problem with the request, a dialog box will appear stating the problem. Correct the problem and click “OK” again.

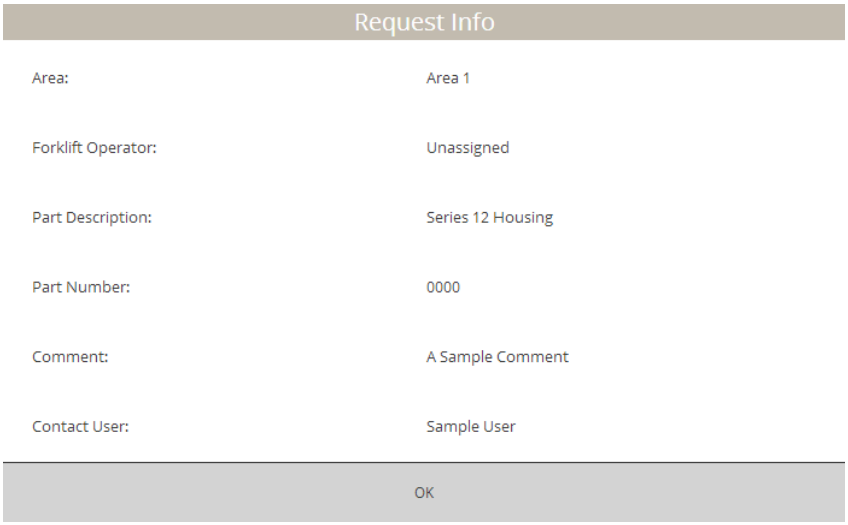
5. The Create Request dialog box will close and the request will appear in the queue.



Note: You may close the “Create Request” dialog box at any time by clicking the X in the upper left hand corner. This will not submit the request and all information in the form will be lost.

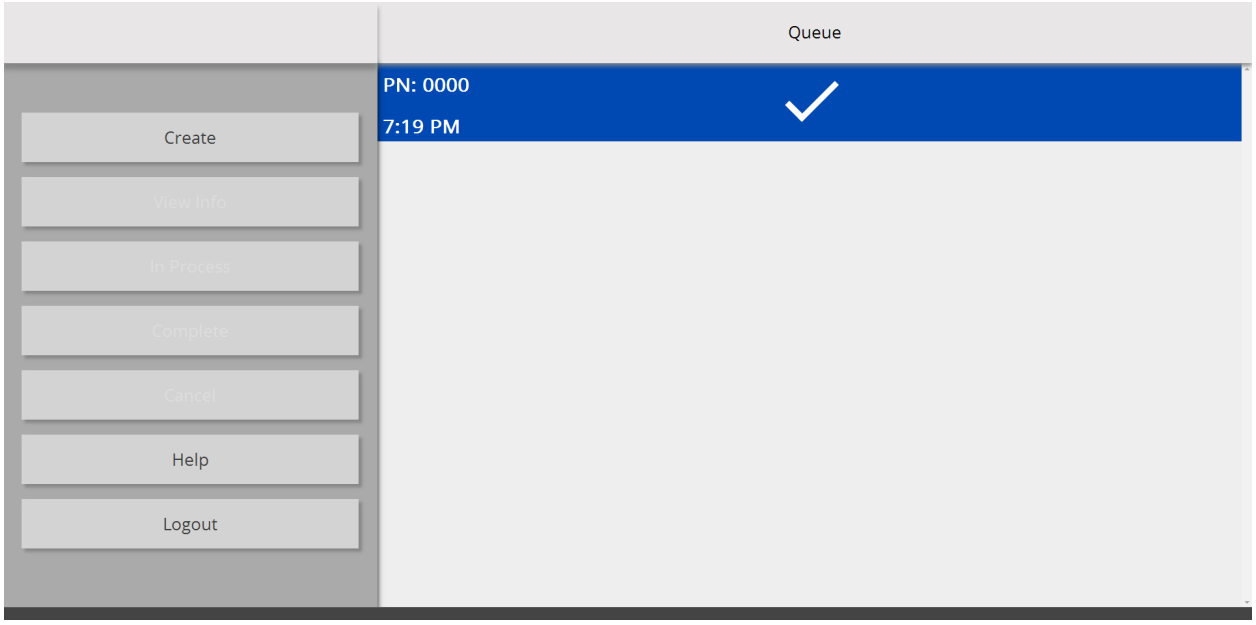
View request info

To view information about a request in your queue, select the request, and click the “View Info” button. Note that the color behind the “Request Info” text at the top; this color is associated with the request’s current state. Press the “OK” at the bottom of the screen to return to the main screen.



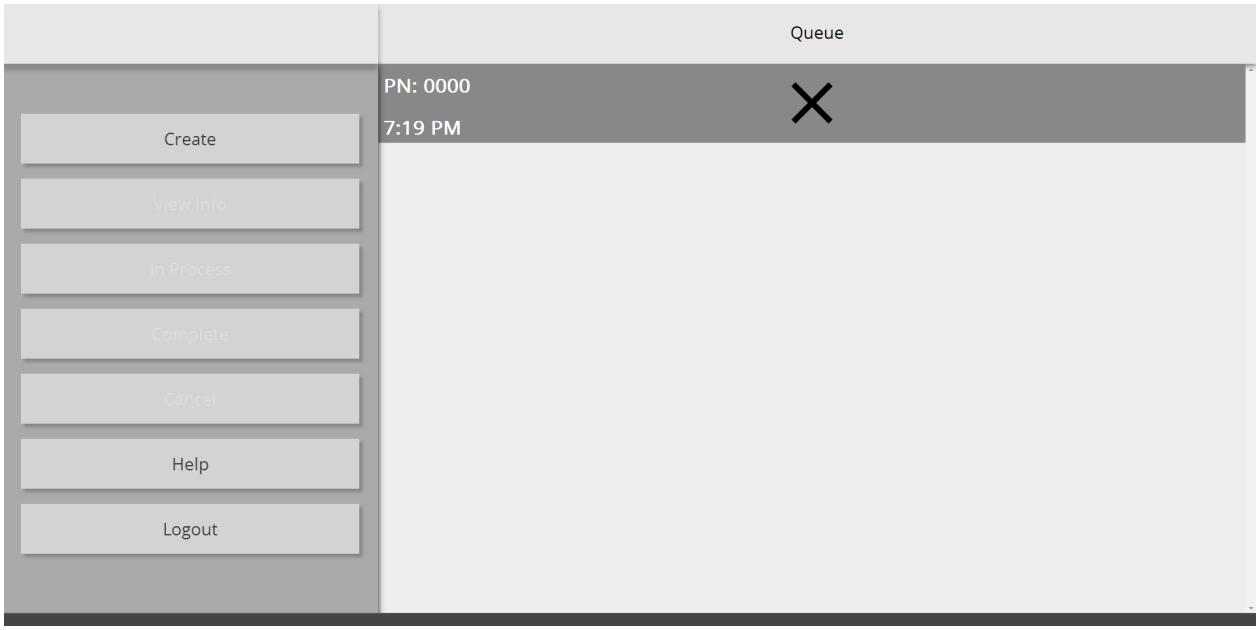
Mark request as completed

To mark a request as complete, select the request in your queue (it will turn light blue when selected), and click the “Mark as complete” button. Note that to make this transition, a request must be in the “in-progress” state. After several seconds, this request will disappear from the queue.



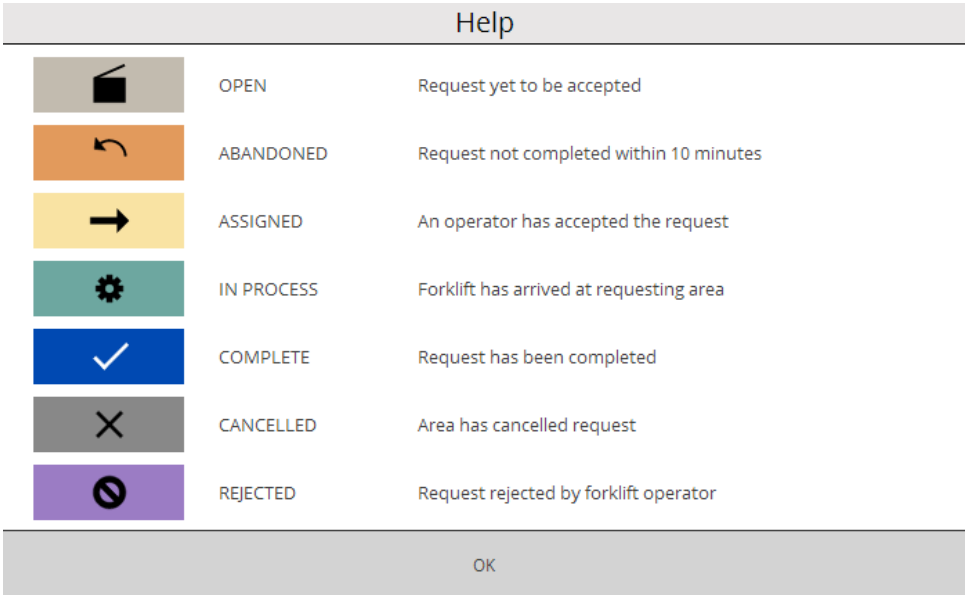
Make request as canceled

If you created a request that is no longer necessary, select the request from your queue, and click the “Cancel” button. Several seconds, this request will disappear from the queue.



View help screen

To get additional help, click the help button at any time. Information about request states can be found on this screen. Press the “OK” button at the bottom of the screen to return to the main screen.



Enter “View only” mode

You can enter a “view only” mode by changing the URL from ending in Area.aspx to ViewOnly.aspx. This will display the main queue without buttons. This mode can be used to display the main queue on large monitors. Press the “OK” button at the bottom of the screen to return to the main screen.

Logout

To logout out, click the “Logout” button.

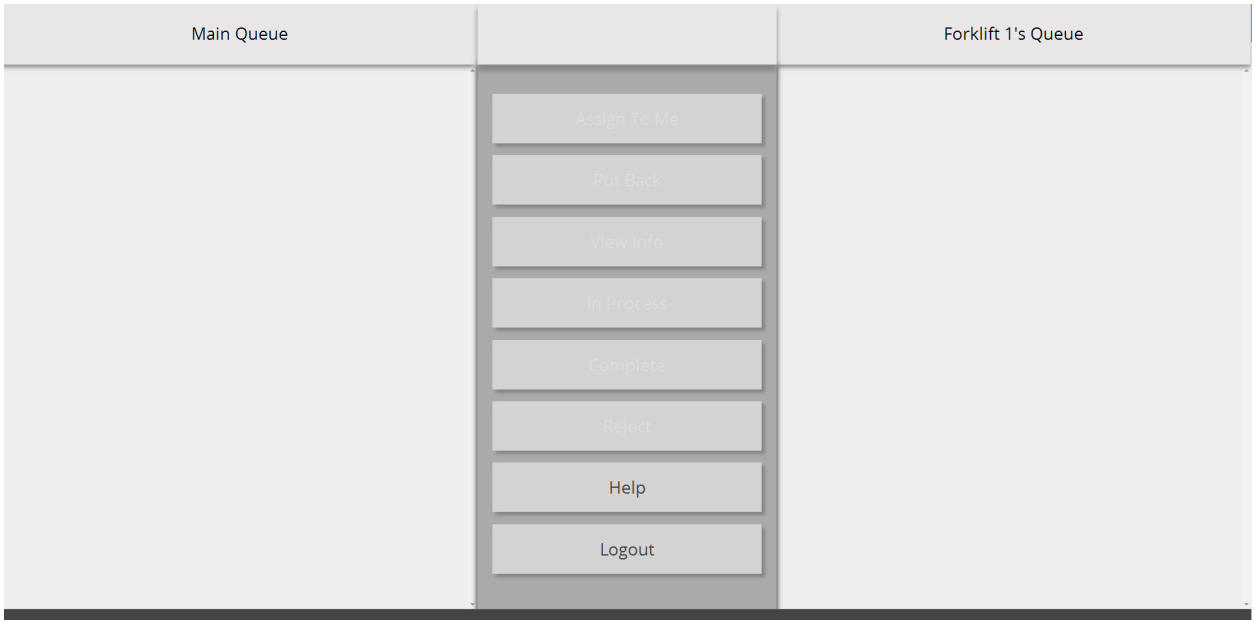
Forklift Operator User

Login

1. Navigate to http://may14-34.ece.iastate.edu

2. Enter your Danfoss assigned username and password and press “Sign In”.

2. If you see a screen similar to the one below, you have successfully logged in. The buttons in the middle show your available actions.



If the application prompts for another login attempt, either you entered your credentials incorrectly, or your username is not configured as a forklift operator. If you encounter this issue, please contact the Danfoss IT Help Desk at x1234. Provide the following information:

- Your Name
- An extension where you can be reached
- The name of the department and/or department number you work for

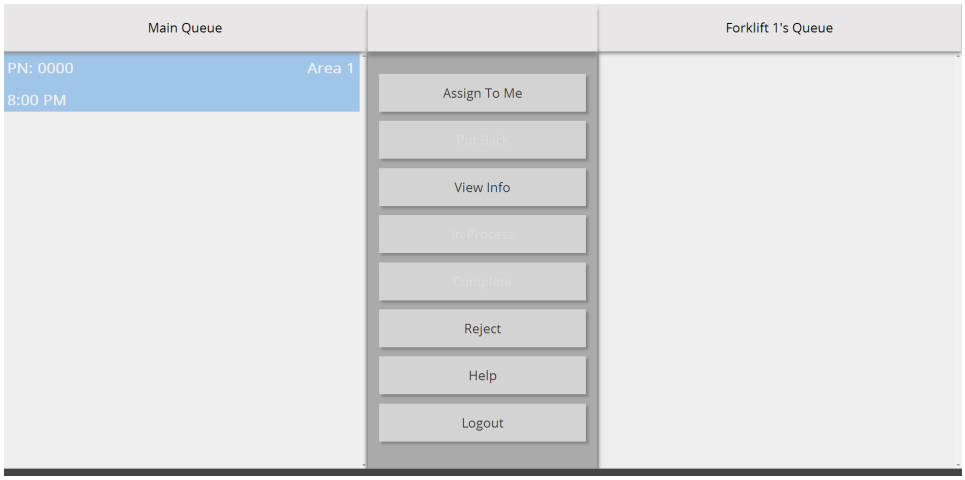
IT will configure your username to access the Forklift Request System as a forklift operator. Once configured, you will be able to use the system.

Understanding Forklift Operator User Commands

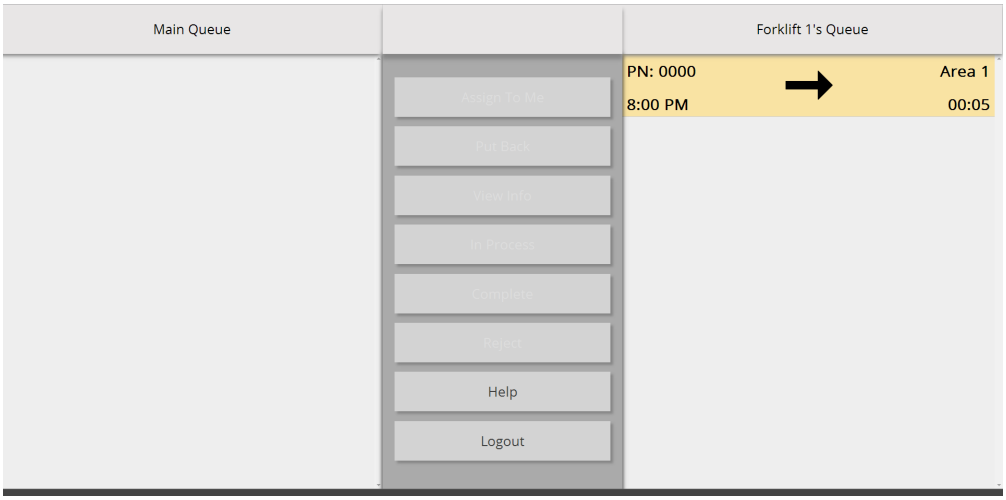
Once you are logged in, there are several commands, some of which may not be available based on the states of the requests in the main queue and your personal queue.

Assign a request

If you are available to service a request from the main queue, select the request by tapping it. It will turn light blue.

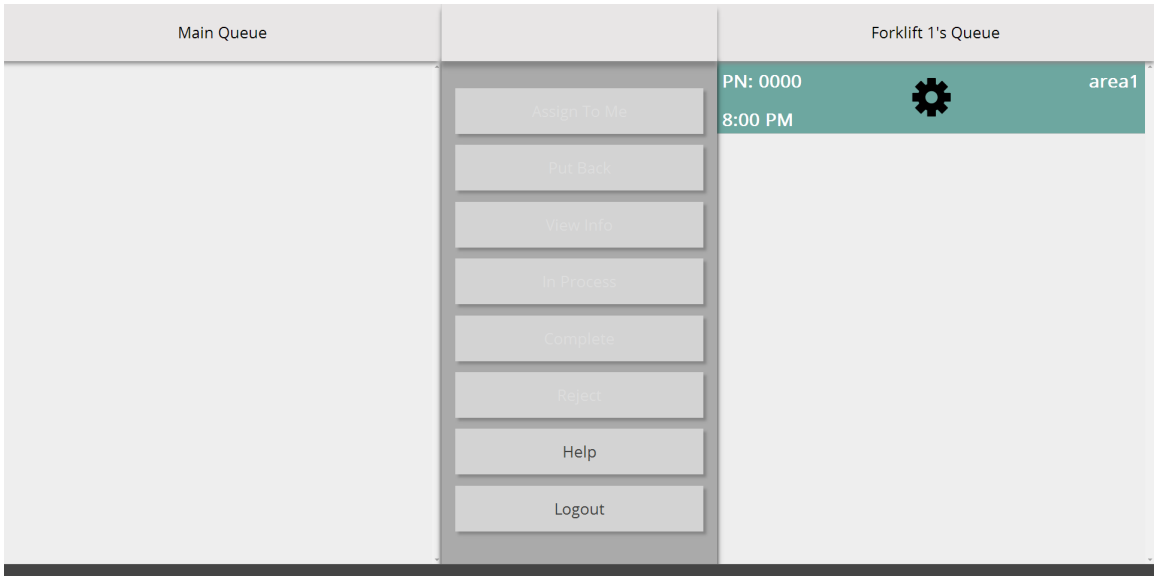


The “Assign To Me” button is now enabled. Press the “Assign To Me” button; the request will be removed from the main queue, and placed at the bottom of your personal queue. If you have many items in your queue, you can scroll up and down to view all your requests. The timer on the left side of a request indicates how long you have to place the request in the “in-progress” state. Otherwise, the request will be marked as “abandoned” and placed back in the main queue.



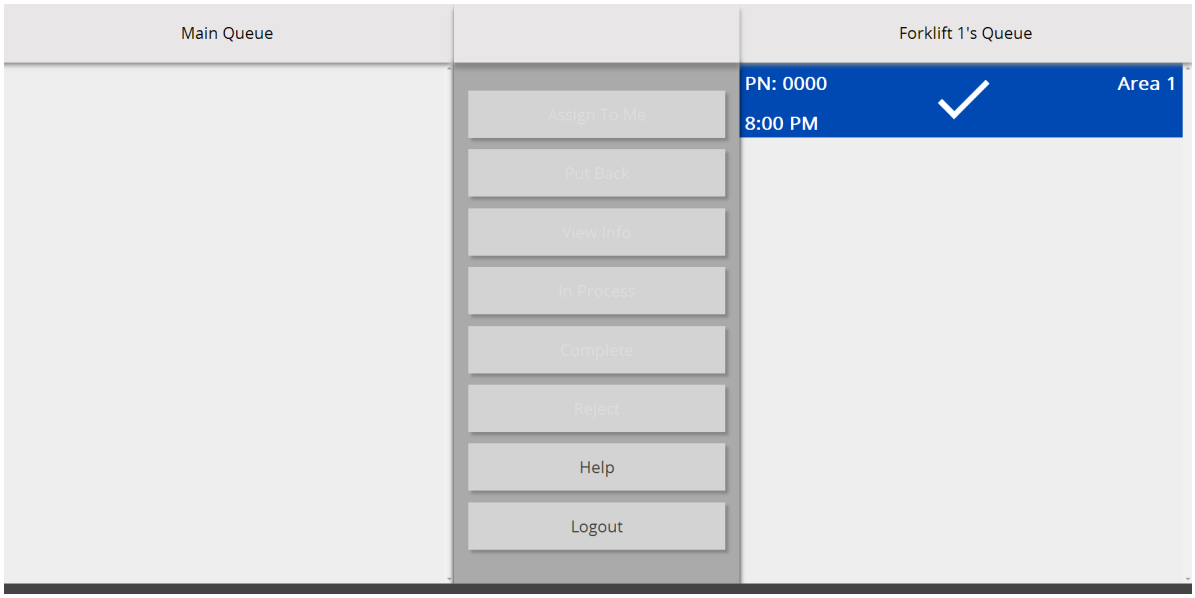
Mark request as in progress

After you have assigned a request to yourself, and you are ready to service it, place the request in the in-progress state. To do this, tap the request; notice the “In Progress” button is enabled. Tap this button, and the request will marked as in progress.



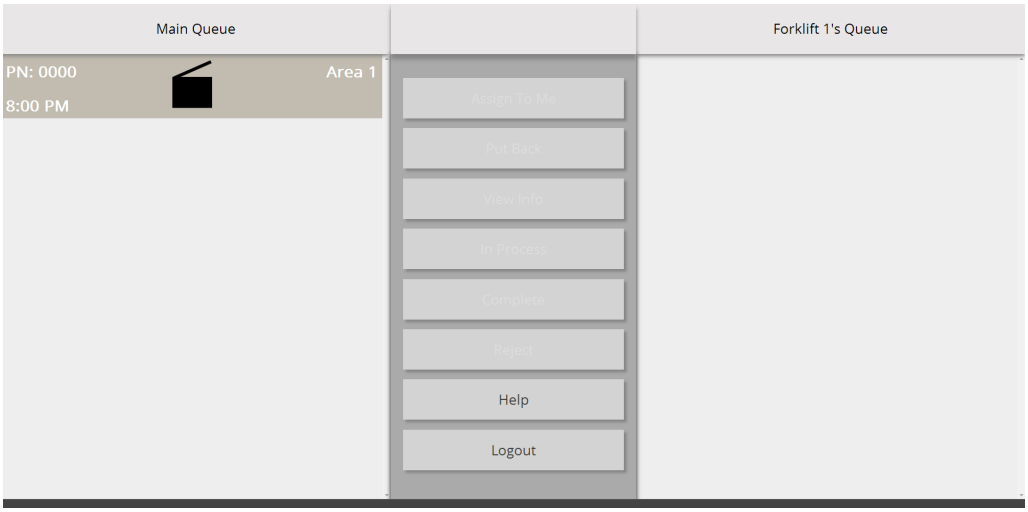
Mark request as completed

After you have completed a request, mark it as complete. Tap the complete request; the “Complete” button is enabled. Tap this button and the request will be set as complete. After several seconds, this request will disappear from the queue.



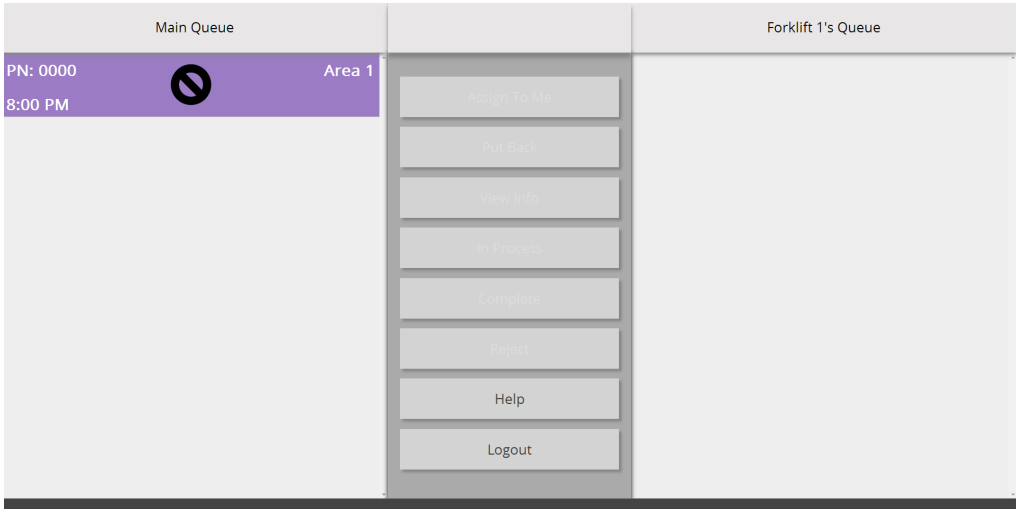
Put request back in main queue

If you find you are unable to complete a request, you should put it back in the main queue so other operators can service it. To do this, select the request; the “Put Back” button will now be enabled. Click it, and the request will leave your queue and return to the main queue.



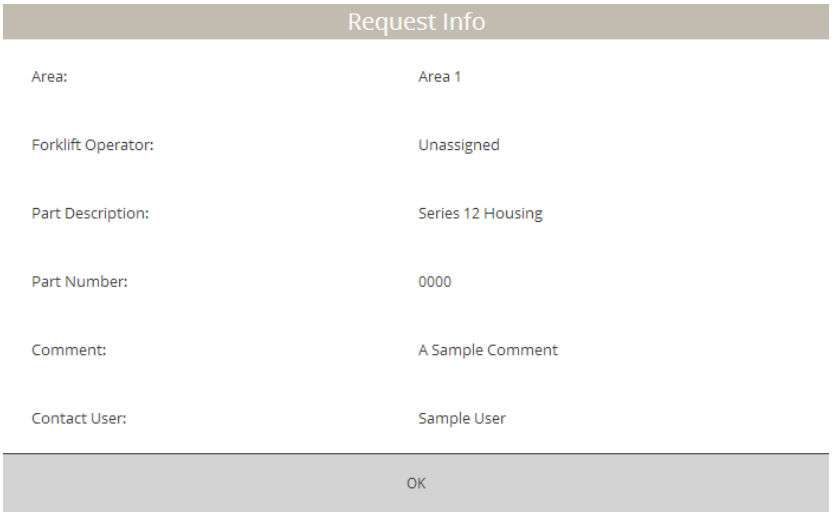
Reject a request

If a request is incomplete or contains incorrect information, you can “reject” it. It is important to note that rejecting a request will remove it from the main queue and notify the request owner. To reject a request, select it; the “Reject” button will be enabled. Press the button, and the request will be rejected. After several seconds, this request will disappear from the main queue.



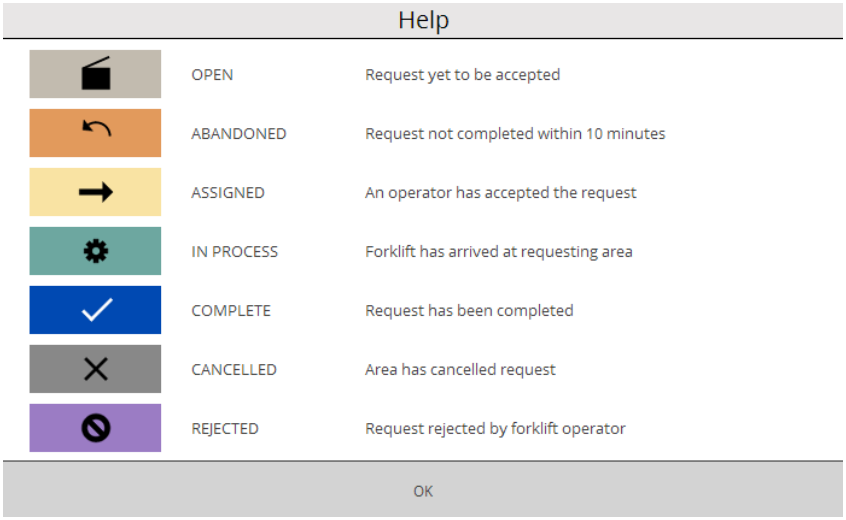
View request info

To view information about a request, select it, and click the “View Info” button. Note that the color behind the “Request Info” text is associated with the request’s state. Press the “OK” button at the bottom to return to the main screen.



View help screen

To get additional help, click the help button. Descriptions about the request states can be found on this screen. Press the “OK” button at the bottom to return to the main screen.



Logout

To logout out, click the “logout” button.

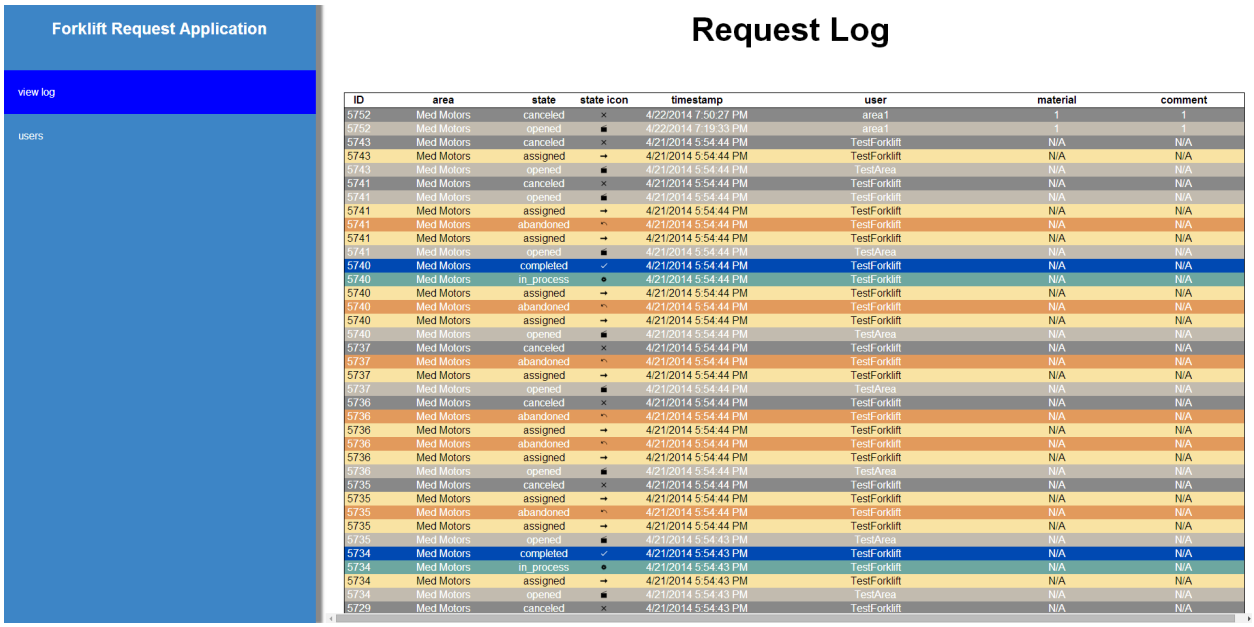
Administrative User

Login

1. Navigate to http://may14-34.ece.iastate.edu

2. Enter your Danfoss assigned username and password and press “Sign In”.

2. If you see a screen similar to the one below, you have successfully logged in. The buttons show your available actions.



If the application prompts for a login, your system is not configured correctly. Please contact the Danfoss IT Help Desk at x1234. Provide the following information:

- Your Name
- An extension where you can be reached
- The name of the PC you are on. This will be of the form AMSPCxxxxx
- The name of the department and/or department number

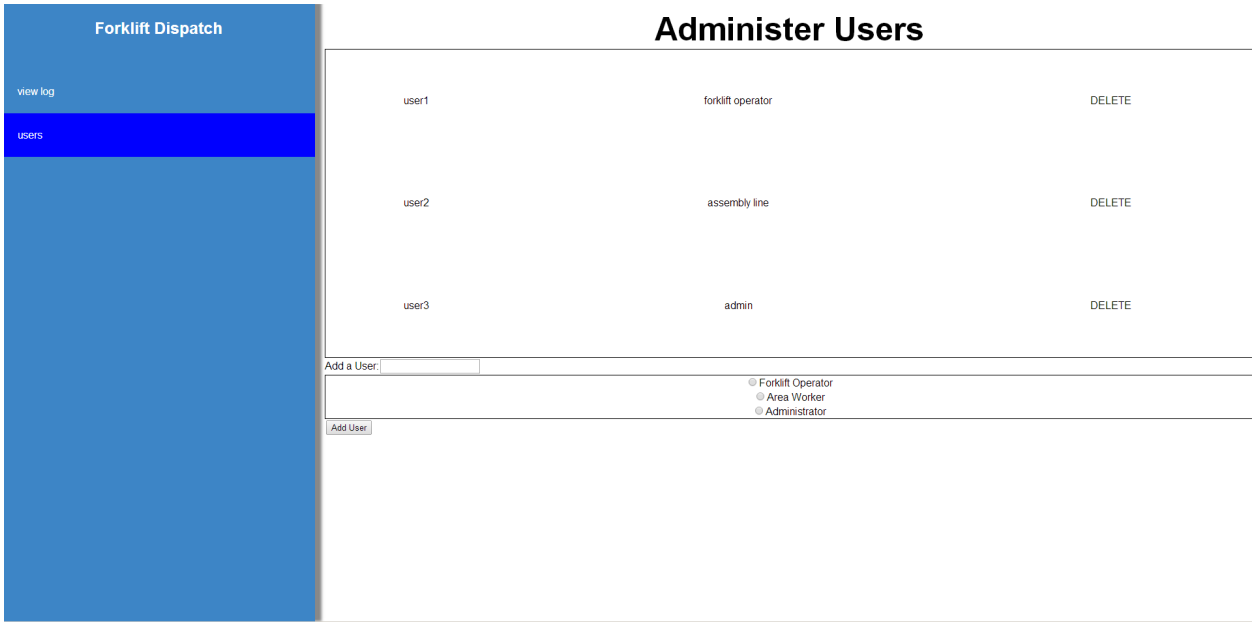
IT will configure the given PC to access the Forklift Request System. Once configured, you will be able to use the system.

Understanding Administrative Commands

Once logged in, administrators can track all requests and manage the Forklift Request Application's users.

Adding and removing a User

On the left menu click "users". This will direct you to the "Administer Users" page, showing a list of all users. To delete a user, select the user, and click "DELETE". To add a user, type their name in the textbox, select their role, and click the "Add User" button. You can see that user added to the user list.



Appendix II: Other Designs

Alternative Database Schema

Initially, two database schemas were proposed. The first held all requests as part of a single table. After careful consideration, two major flaws with this schema were discovered. First, the number of rows in the table would be large: the application is "read" heavy, and thousands of rows would hinder the performance of the application considerably. Second, if each request had its own row, it would be impossible to *fully* track the state transitions. Since this type of tracking was desired by Danfoss, the second schema was selected and implemented.

"Pushing" Data from Server to Client

The team considered several methods for the server to "push" events to all connected clients (web browsers). The application was constrained (as a web application) to run over HTTP Protocol. HTTP is a "pulling" protocol or "half-duplex". This means the client asks the server for data, and the server *cannot* initiate data transfers to the client.

This issue can be illustrated from the following example: a client creates a Forklift Request and informs the server. The server performs the action and returns "success" to the client. How do other clients know about this new request? Remember, the server cannot "push" this event to them.

There are many solutions for this issue. First are browser plugins, such as Adobe Flash and Java. These plugins allow "full-duplex" communication, meaning the server *can* "push" data. However, browser plugins add overhead, complexity, and dependency, and were not seriously consider for this application.

A second solution is for each client to continually ask the server for new events. This is called polling. Usually, the server responds "no", and this solution is bandwidth heavy, inefficient, and introduces lag to the system.

A slight variant of the second solution is called "Long-Polling" or Comet. Similar to polling, the client continuously asks the server for more data, but in this case, the server keeps each connection open until a new event has occurred. Upon a new event, the server responds to the request. Then, the client reconnects and the process starts again. This solution removes the lag and overhead mentioned above, but still requires the client to continually call the server.

A third solution is a new protocol called WebSockets, defined in RFC 6455[1], which was finalized in 2011. WebSockets allow full-duplex communication, however they are not supported in Internet Explorer 8, which Danfoss requires us to support. Due to this restriction, WebSockets could not be used as a solution.

[1] https://tools.ietf.org/html/rfc6455

After careful consideration of each solution, Long-Polling was selected for the final design and implementation. Long-Polling best fits the needs of the application and its functional constraints.

Alternative JavaScript Libraries

As discussed in *Implementation Details*, the Frontend is written in JavaScript, which is supported on all modern browsers. Unfortunately, various web browsers will execute the same piece of JavaScript code slightly differently. In other words, each browser has its own "dialect". A naive solution is to write multiple versions of the Frontend--one for each supported browser. This results in duplicate code and creates maintainability issues.

Several JavaScript Libraries exist to address this issue. These libraries take native JavaScript and "translate" it to each browser dialect. This allows developers to write once, and have code that works uniformly across all browsers.

Some libraries include: MooTools[2], jQuery[3], and SyncFusion[4]. Each have benefits and drawbacks. Because Danfoss uses jQuery, and the team was already familiar with it, jQuery was chosen as the JavaScript library for this application.

Client Side Request Timeouts

As discussed previously, if a request is assigned to a forklift operator past an administrative defined threshold, the request is automatically set to the abandoned state. The implementation has an "abandoned checking" service, which runs continuously on the server. When the service marks a request as abandoned, the server informs all connected clients.

The team considered another implementation, which would have the client keep track of the abandoned state. Given the risk of users logging out of the application or losing network connection, this alternative was deemed unacceptable.

Send All data or just new data?

When a user is navigating a traditional web-page and wants new data, he refreshes the page and the server sends the entire page again. This method is simple, but less efficient than just sending new data. The team decided to go with the second method (only new data) to reduces server, network, and client load--especially when there is a large number of requests in the database.

How Assembly Line Employees "Login"

Originally, all users were going to login via a Danfoss assigned username and password. The application would look up the role associated with the username (i.e. assembly line employee, forklift operator, or administrator) and redirect the user to the appropriate view. Later, Danfoss

[2] http://mootools.net/
[3] http://jquery.com/
[4] http://www.syncfusion.com/

pointed out that Assembly Line employees always use the same workstation. Because of this, Danfoss requested Assembly Line employees be automatically logged-in based on their station's unique computer identification number.

Six Request States

The original design called for six request states. In January 2014, Danfoss requested the addition of a rejected state. Because of the modular design of the application, the change was trivial.

Appendix III: Other Considerations

During the design phase of this application (EE/CprE/SE 491) other considerations and additions to the application were suggested but never incorporated into the design or implemented. Time, team knowledge, and Danfoss' priority list are all valid reasons as to why the following considerations never went beyond brainstorming.

HTML5 Analytic Graphs

Initially, it was planned that the application would be able to create detailed graphical analytics over the growing forklift request dataset. Graphs such as "Average Completion Time" and "Average Time Spent In-Progress" would be generated on-the-fly and presented to administrators/managers at Danfoss via a webpage.

Danfoss already uses a graphical reporting solution called SSRS (SQL Server Reporting Studio). After a few discussions with them, it was decided that creating the graphs from scratch would be "reinventing the wheel" and add unneeded complexity to the application.

In place of the analytic graphs, SSRS reporting queries (which pull relevant data from the application's database) were created instead and given to Danfoss. All Danfoss has to do is import those queries into the SSRS program and they can create all the graphs they need to suit their needs.

Allowing the Addition of Request States

Initially, six request states existed with the plan for administrators to be able to add request states via the administration screens of the application. In January of 2014, Danfoss wished to include an additional request state: "rejected". The application design is modular and allowed for this addition with ease, however it was not as simple as adding the state to the database. For this reason, it was decided that adding states on the fly could not be an option.

Allowing the Addition of User Roles

Initially, as part of project plan v1.0, adding user roles via the administration portion of the application was to be allowed. Given the application behaves differently based on the currently logged in user, adding a user role on the fly would not work. A completely different "view" of the application would have to be created. While this would be easy to accomplish, it would require development and a redeployment of the application. This means the developer modifying the application could add the user role instead.

Auto Assigning of New Requests

A “feature” that never even made it to project plan v1.0 was auto assigning a new request to the “least” busiest forklift operator. While the algorithm to complete this task would be relatively simple, when suggested to Danfoss, it was immediately decided this would not be an incorporated feature. Danfoss wanted the employees to be in complete control of what was going on with very little artificial intelligence.

Providing a Map on the Forklift Operator’s Screen

Originally it was planed the application would place a detailed map of the Danfoss factory on each forklift operator’s view. The map would display the forklift operator’s current position and the position of the item he was going to retrieve. When presented to Danfoss, the idea was described as overly complicated, error prone, and the algorithm would never be as intelligent as an experienced forklift operator.

“Stacking” Request Timeouts

A “feature” never implemented or part of the design plan was “stacking” request timeouts. Basically, when a forklift operator assigns a request to himself, a timer starts on that request. If the operator does not complete that request before the timer runs out, then the request leaves their queue and goes into the abandoned state. This timeout is set to a constant value. “Stacking” these timeouts instead means setting the timeout higher based on the number of requests in the forklift operator’s queue. For example, assume the timeout constant is set to 5 min. Now consider that same forklift operator has a request in their queue with 2 min. left. If he assigns another request to himself, the timeout for that request would be set to 7 min. rather than 5 min. had “stacking” request timeouts been part of the design or implemented.