May15-05 Design Document

A.R.T.I.M.U.S
Augmented Reality Tractor Information Management and Utility System

For John Deere Intelligent Solutions Group

Jesse Walther, Team Lead
Han Sang Youn, Testing Lead
Haoyu Liu, Webmaster
Brian Moran, Key Concept Holder
Tanner Hildebrand, Communications Lead

Manimaran Govindarasu, Advisor

Contents

Figures and Tables

Figures

Document Updates

Document Action	Date Complete
Document Birth (TH)	1 SEP 2014
Version 1 Prepared for Review – Use Cases, I/O, Hardware, Modelling, Testing, Summary/Conclusion (TH)	15 OCT 2014
Addition of Application specific system Diagrams (TH)	25 OCT 2014
Version 2 Prepared for Review – Updates to System Design and Diagrams (TH)	1 DEC 2014
Appendixes Added (TH)	7 JAN 2015
Updated Parameters from John Deere for functional requirements (TH)	1 FEB 2015
Developer Manual Updates (TH)	1 MAR 2015
Final Document Preparation (TH)	21 APR 2015

Table 1: Document Updates

Project Introduction

Project Statement

Currently the John Deere API can serve some maintenance information on their tractors from their server that has been downloaded from the tractors connection in the field. However, a farmer in the field may not have connection to cellular or wireless service needed to reach the John Deere servers; this application is intended to fill that gap by connecting directly to the tractor via wireless to a device on board the tractor.

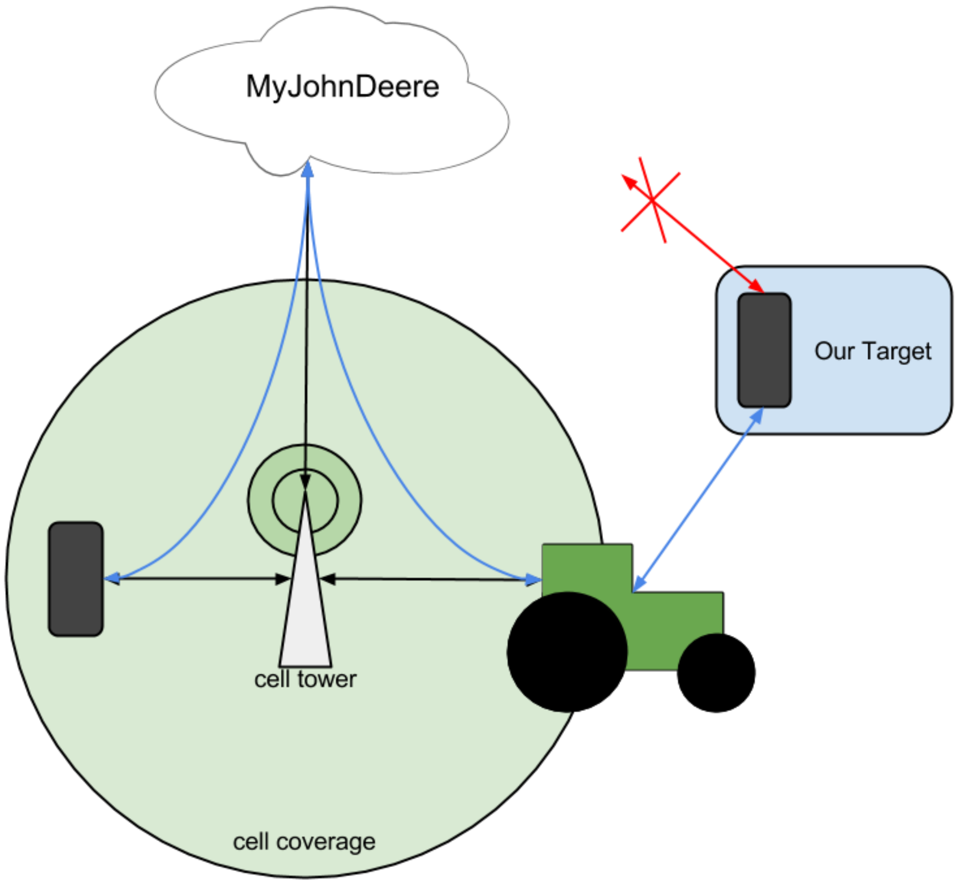


Figure 1: Tractor Connection Problem

Deliverables

The final product for this project will be a mobile application for the Apple iOS platform which runs on iOS 7 and up.

A prototype will be available for demonstration at the end of the first Semester to show to ISG and will work as a proof of concept for future development.

Every month a demo will be available to show ISG and separate tag in the git repository will be created to benchmark these demos.

Specifications

This application provides the following

Key Functional Requirements

- Connects directly to MTG device on board the tractor via wireless and downloads maintenance data securely to the mobile device.
- Augmented Reality View—provides a point and shoot method to identify a tractor and display maintenance information on this tractor.
- Maintenance Information and Alerts—provides information on the maintenance state of the tractor and will display alerts when maintenance parameters pass pre-defined thresholds.
- The application will work in the following 3 modes:
    - Farmer Mode—provides all the features listed above.
    - Dealer Mode—provides all the features of the farmer mode and in addition this mode will allow a dealer to quickly sort, classify, tag, and search for vehicles he has registered on his app.
    - Developer Mode—provides all the features of the dealer mode and in addition the app will be able to download logs from the MTG and may provide some information about these logs.

The application will track the following maintenance information and provide alerts as needed for (continued on next page):

Combine Model S690

Parameter to be displayed if user is a Farmer

Parameter	Create an Alert if value is above (or below) this value
Fuel Level [Capacity]	Less than 65 gallon [Capacity is 330 gal]
Machine Hours	Above 50 hrs. Also display total machine hours each time.
Coolant Temperature	Above 85C [if above 95C, engine stop code is generated]
Hydraulic Oil Temperature	55C
RDA License Available	NA
WDT License Available	NA
Machine Sync Available	NA
GPS Working	No GPS info for more than 5 min.
Connected Devices	NA
Tire Pressure	If less than 160kPa [recommended value is 193kPa, 28psi]

Table 2: Combine Model S690 Farmer Parameters

Parameter to be displayed if user is a Dealer

Everything that is displayed to a Farmer

Parameter	Create an Alert if value is above this value
MTG Serial Number	NA
Last Cell Call In time	More than 5 days.
Last reprogramming status	Alert if last reprogramming attempt failed.
MTG Firmware Version Number	NA

Table 3: Combine Model S690 Dealer Parameters

Parameter to be displayed if user is a Developer

Everything that is displayed to a Dealer and MTG Debug Logs

Sprayer Model R4038

Parameter to be displayed if user is a Farmer

Parameter	Create an Alert if value is above (or below) this value
Fuel Level [Capacity]	Less than 40 gallon [Capacity is 155 gal]
Machine Hours	Above 50 hrs. Also display total machine hours.
Coolant Temperature	Above 85C [if above 95C, engine stop code is generated]
Hydraulic Oil Temperature	Above 88C
RDA License Available	NA
WDT License Available	NA
Machine Sync Available	NA
GPS Working	No GPS info for more than 5 min.
Connected Devices	NA
Tire Pressure	If less than 420kPa [recommended value is 441kPa, 64psi]

Table 4: Sprayer Model R4038 Farmer Parameters

Parameter to be displayed if user is a Dealer

Everything that is displayed to a Farmer

Parameter	Create an Alert if value is above this value
MTG Serial Number	NA
Last Cell Call In time	More than 5 days.
Last reprogramming status	Alert if last reprogramming attempt failed.
MTG Firmware Version Number	NA

Table 5: Sprayer Model R4038 Dealer Parameters

Parameter to be displayed if user is a Developer

Everything that is displayed to a Dealer and MTG Debug Logs

Tractor Model 8R- with Group 47/48/49 Tires

Parameter to be displayed if user is a Farmer

Parameter	Create an Alert if value is above (or below) this value
Fuel Level [Capacity]	Less than 50 gallon [Capacity is 190 gal]
Machine Hours	Above 50 hrs. Also display total machine hours.
Coolant Temperature	Above 85C [if above 95C, engine stop code is generated]
Hydraulic Oil Temperature	55C
RDA License Available	NA
WDT License Available	NA
Machine Sync Available	NA
GPS Working	No GPS info for more than 5 min.
Connected Devices	NA
Tire Pressure	If less than 70kPa [recommended value is 80kPa, 12psi]

Table 6: Tractor Model 8R Farmer Parameters

Parameter to be displayed if user is a Dealer

Everything that is displayed to a Farmer

Parameter	Create an Alert if value is above this value
MTG Serial Number	NA
Last Cell Call In time	More than 5 days.
Last reprogramming status	Alert if last reprogramming attempt failed.
MTG Firmware Version Number	NA

Table 7: Tractor Model 8R Dealer Parameters

Parameter to be displayed if user is a Developer

Everything that is displayed to a Dealer and MTG Debug Logs

The following additional features are expected.

Non-functional Requirements

- Application will run on iOS 7 and up.
- This application must make use of best practices in order to avoid transmitting sensitive data over unsecured networks and take steps to protect any data that does pass over a network through any medium.
- The applications operation must not expose or reveal any sensitive information to anyone but an authenticated user..

System Design

For a more detailed description of how the application is implemented refer to Appendix A:o��o�Appendix B Developer Manual below.

System Requirements

This application will run on Apple's proprietary iOS mobile operating system starting from version 7 (iOS 7) and up (current highest version is iOS 8). This application is intended for use on iPhones running appropriate OS versions.

Devices using this application should have a working camera to use any Augmented Reality or image recognition features to identify the vehicle.

In order for this application to be functional WiFi connectivity must also be available on the device and enabled so that the device may connect to the MTG on board the tractor to communicate.

System Analysis

Feasibility

There are several resources available which make this project very feasible:

Augmented Reality:

We will likely use Vuforia as a framework for doing Augmented Reality. It is free to use and offers options for developers to use their image recognition for free. Additionally their service widely accepted, used, and documented which will aid our development process to no end. This also handles much of the Camera and Image Recognition requirements for this application.

iOS Development:

Xcode is readily available in the TLA in Coover on ISU's campus. Development for iOS on Xcode is greatly facilitated such that learning Objective-C for development is made simple and feasible.

External APIs

The MTG's API will be provided by ISG which we expect will be well supported as we will be working directly with it. Thus we expect no issues with this. The MyJohnDeere API is documented on its resource site and appears to provide all necessary information at this time. Vuforia's API is widely used and well documented; Vuforia is expected to be the most difficult to find support for but due to its popularity does not appear to pose a problem.

System Block Diagram

Current versions of the MTG do not have a built in WiFi chip but it did have an ethernet port. Using a TPLink device configured to work as an Access Point and a Gateway for the MTG we could establish an individual connection. Future versions of the MTG may have native WiFi capabilities.

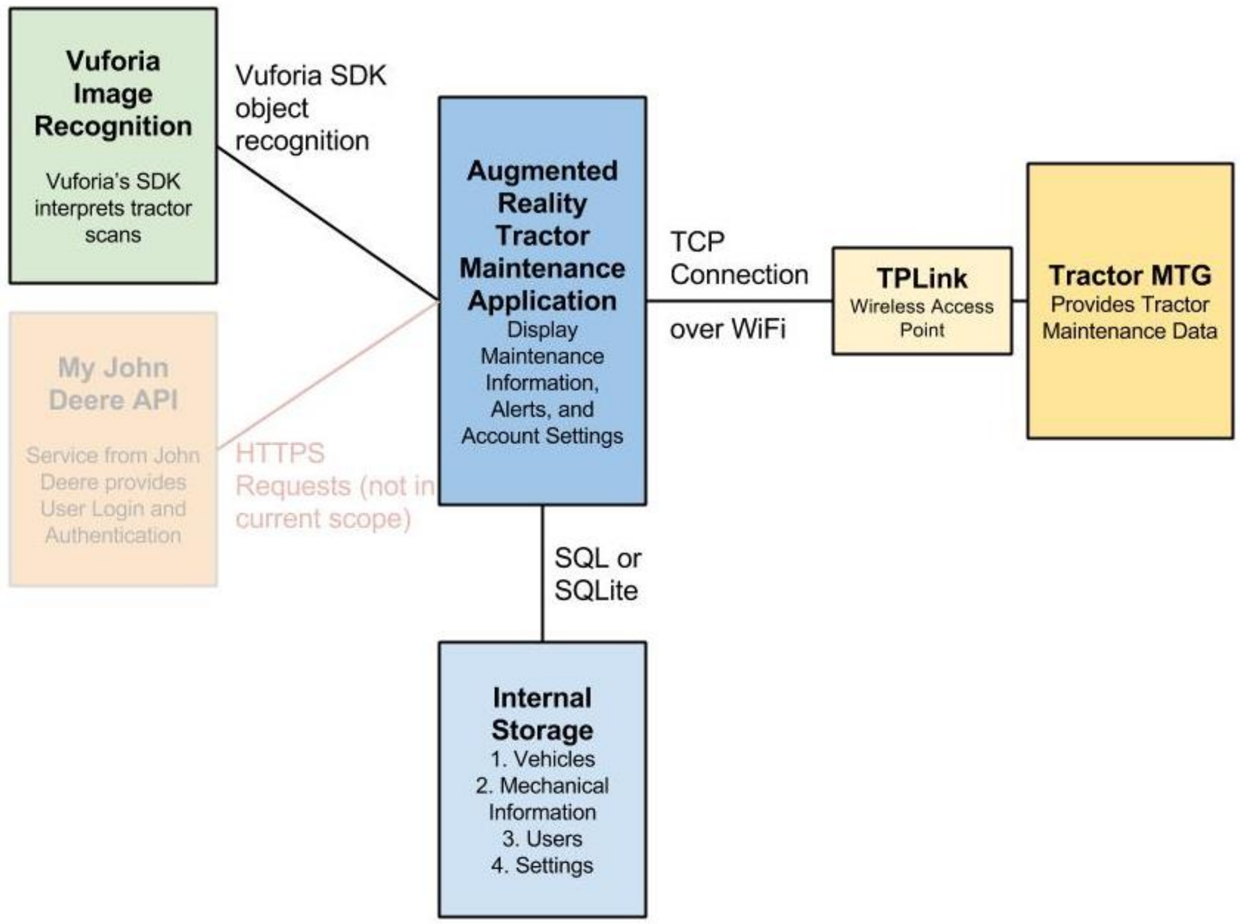

Vuforia Image Recognition
Vuforia's SDK interprets tractor scans
Vuforia SDK object recognition
My John Deere API
Service from John Deere provides User Login and Authentication
HTTPS Requests (not in current scope)
Augmented Reality Tractor Maintenance Application
Display Maintenance Information, Alerts, and Account Settings
TCP Connection over WiFi
TPLink
Wireless Access Point
Tractor MTG
Provides Tractor Maintenance Data
SQL or SQLite
Internal Storage
1. Vehicles
2. Mechanical Information
3. Users
4. Settings

Figure 2: System Block Diagram

Application Block Diagram

Application flow relies on the Apple iOS navigation stack which contributes well to ease of use in our application and a natural, native feel for users. Where ever possible the design adheres to Apple's iOS design guidelines.

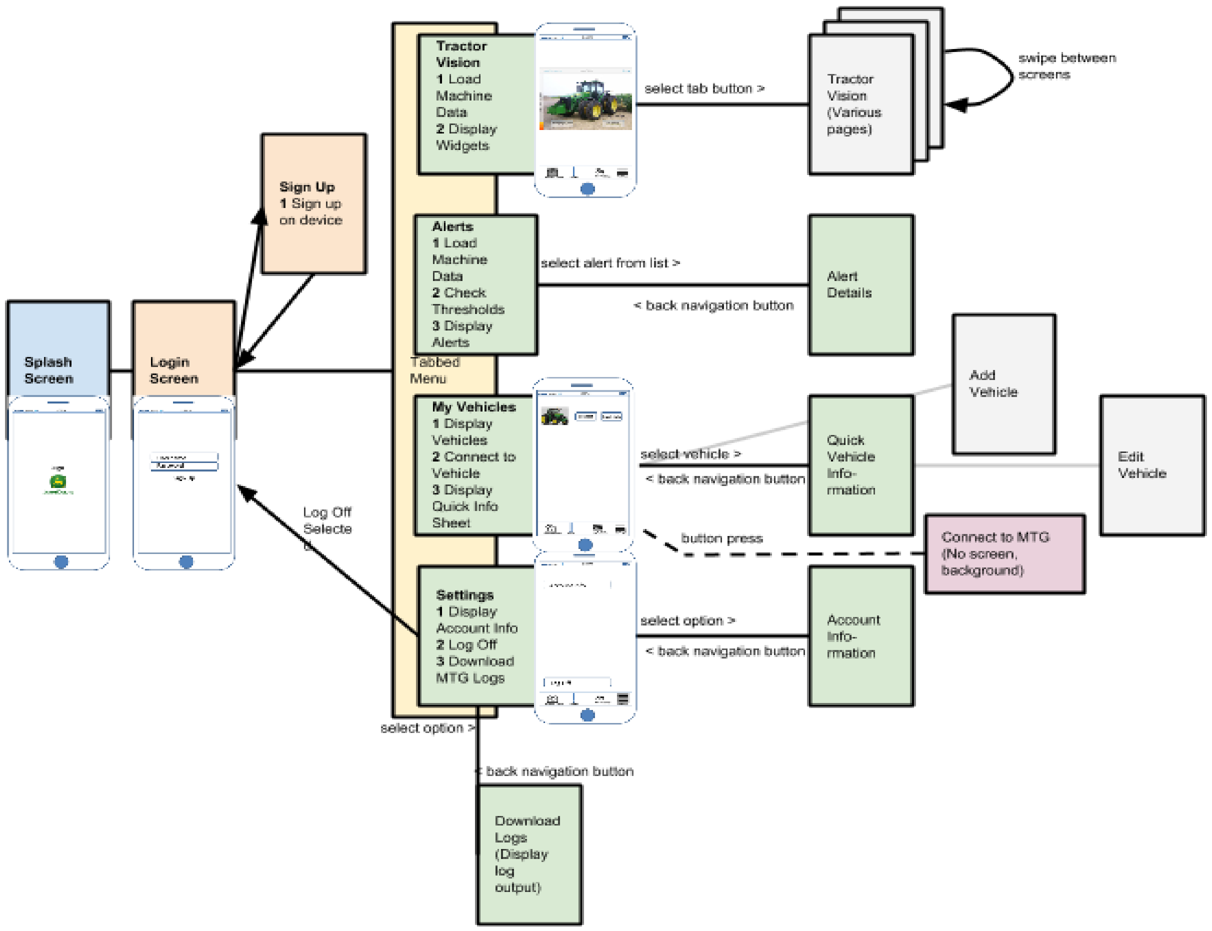


Figure 3: Application Screen Flow

Functional Decomposition

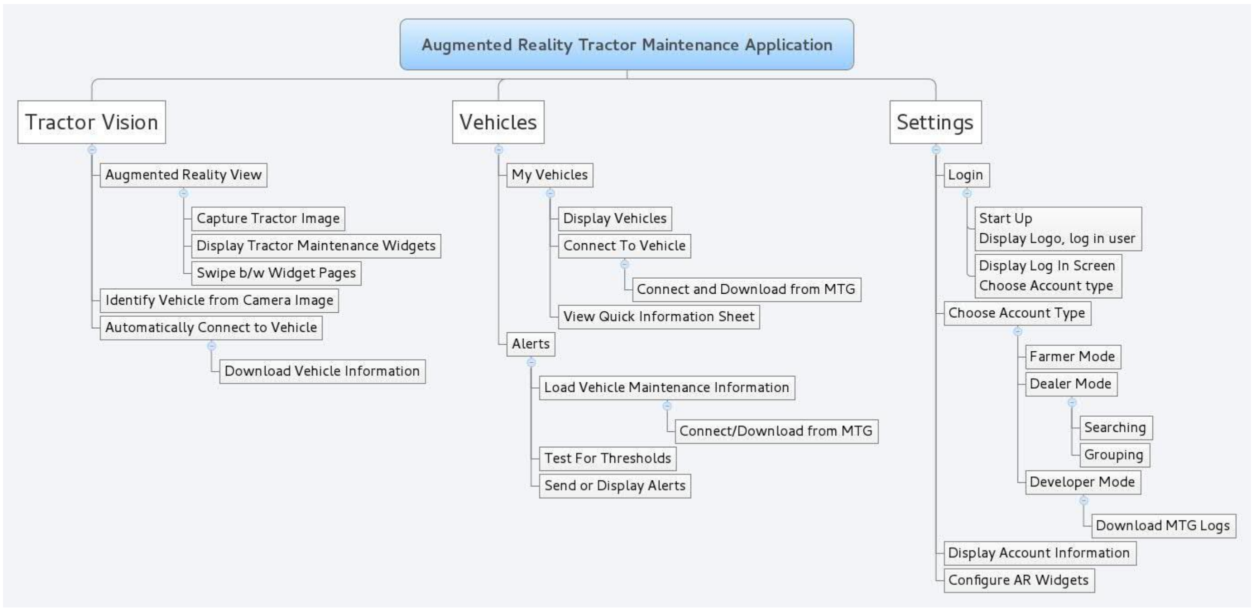


Figure 4: Functional Decomposition

Use Cases

Use Case 1: User Logs In

Actor: Application user

Primary flow:

User starts the application and is presented with a splash screen for a 2 seconds before the log in screen is presented. User enters username and password and chooses log in. User is validated using MyJohnDeere and is brought to the default screen. The application records the login status.

Alternative flow 1: User log in fails

Instead of being brought to the default screen user is presented with a message that login has failed and is brought back to the log in screen to resume primary flow.

Alternative flow 2: User chooses 'Sign Up'

User is transferred via http link to the MyJohnDeere sign up page.

Alternative flow 3: User is already logged in

If the user has already logged into the application and the application remembers the users data, the user will skip the login screen and transfer directly to the application screen.

Use Case 2: View Vehicles

Actor: Application user

Primary flow:

User chooses the 'My Vehicles' tab button and is presented with a list of tractors which are registered to their account.

Alternative flow 1: User selects to view info on the tractor

If the user selects the tractor they will be transferred to the Tractor's Quick Information page with details about the tractor's registered information.

Alternative flow 2: user Selects 'Connect' button

The Application will attempt to connect to connect the tractor via TCP connection over wireless. If successful an alert will be shown and connect will be disabled, if not successful an alert will be shown and button will still be enabled.

Use Case 3: View Settings

Actor: Application user

Primary flow:

User chooses the 'Settings' tab button and is presented with a list of settings options.

Alternative flow 1: User selects account information

Information about the user account with MyJohnDeere is displayed.

Alternative flow 2: User selects application information

Options specific to the application are displayed here, used for debugging and checking version number, etc.

Alternative flow 3: User selects log out

The user's status is changed to logged out and the application is brought back to the login screen.

Use Case 4: View Alerts

Actor: Application user

Primary flow:
User selects the 'Alerts' tab button and is brought to the alerts page. The application checks parameter values in data downloaded from the MTG against a list of thresholds and displays alerts if they are present.

Alternative flow 1:
User selects an alert and is brought to a page with details about the alert and recommendations on how to fix.

User Case 5: View Augmented Reality
Actor: Application user

Primary flow:
User selects 'Tractor Vision' tab button and is brought to the Augmented Reality screen view. The camera is started and begins to try and recognize a tractor on the screen. If it recognizes a tractor it will attempt to display information on that tractor with overlaid widgets.

Alternative flow 1: User freezes the screen
If the user taps on the screen they may freeze the screen so they do not have to hold the camera up constantly. All widgets stay present but the camera no longer shifts.

Alternative flow 2: User swipes through screens
If the User swipes across the screen they may view different screens containing different data widgets much like a home screen with application icons/widgets.

Detail Description

I/O Specifications
This device communicates primarily through wireless with a device on board the Tractor called an MTG.

In order to identify and connect to a machine the user must connect to the TPLink device connected to the MTG as an access point. TCP connection is then established to send requests to the tractor. The protocol for downloading data is simple involving a simple connection and minimal-to-no handshaking. The functions and services on the MTG will be provided by ISG through these TCP requests.

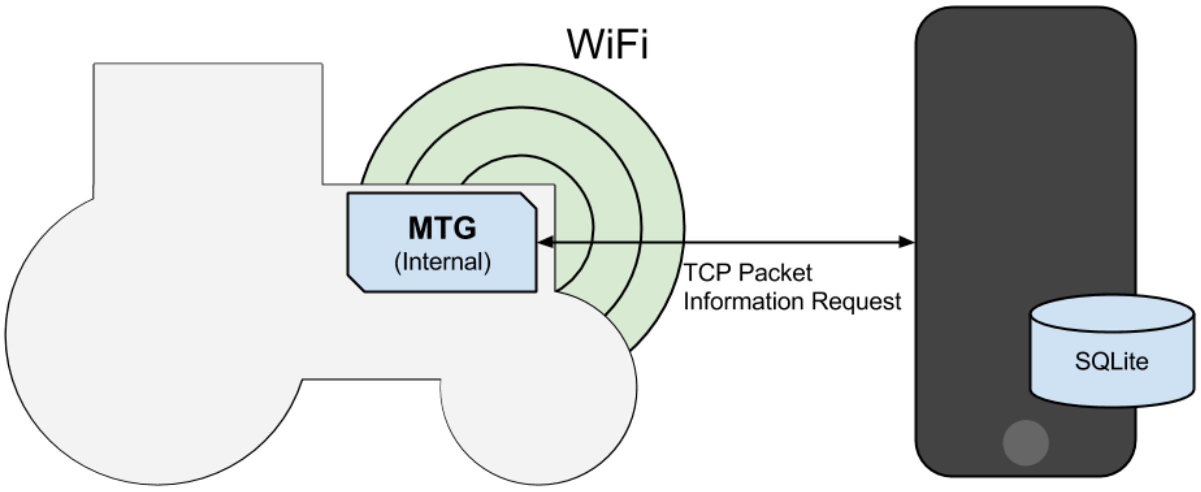


Figure 5: Basic Communication

The connection protocol looks like this:

Application		**MTG**
Set up:		
Connect to AP for vehicle		
Connect:		
TCP Connection request to 172.16.0.2 port 45544	→	MTG accepts connection
Receive data structure	←	Immediately transmit maintenance data structure
Finish:		
Close Connection	(both sides)	Close Connection
Parse data structure		Await new connections

Table 8: Communication Protocol

Other I/O

The application will also make use of the camera to capture a tractor. The camera features are to be handled through the Apple camera framework provided with the iOS SDK.

Interface Specifications

Primary Application Modules

This application does not provide an API for use outside of the application. For information on internal interfaces see Appendix A.

John Deere API

While not in the current scope it is possible that future versions of the application will interface with the MyJohnDeere API in order to provide user log in and authentication. This will all be done over HTTPS in order to maintain user information security and privacy.

https://developer.deere.com/#/home/landing

Vuforia API

This application will make use of the Vuforia Augmented Reality software framework/library for iOS. The recently released version (4.0) for 64 bit iOS will be used for our system.

https://developer.vuforia.com/

MTG API

See more on this in the Communication section.

Hardware Specifications

Mobile Device Specifications

The application will support only the standard Apple iPhone devices running the appropriate iOS version. As such the hardware supported will only be defined to be authentic factory Apple iPhones from iPhone 4 and up.

See these references for further information on the iPhone hardware specifications.

iPhone 4:

http://www.everymac.com/systems/apple/iphone/specs/apple-iphone-4-verizon-cdma-specs.html

iPhone 5:

http://www.everymac.com/systems/apple/iphone/specs/apple-iphone-5-a1429-cdma-lte-sprint-verizon-specs.html

iPhone 6:

http://www.everymac.com/systems/apple/iphone/specs/apple-iphone-6-a1549-4.7-inch-cdma-verizon-north-america-specs.html

List of all recent Apple iPhone products (for other specific versions, i.e. 5s, 5c):

http://www.everymac.com/systems/apple/iphone/index-iphone-specs.html

MGT Specifications

The current version of the MTG runs Windows CE and does not have a WiFi chip on board however we are able to connect a TPLink device to the MTG which can be configured to work as an access point and a gateway for the MTG. With this we are able to join the TPLink's network and connect to the MTG which has an Ethernet port and a hardcoded IP address.

Future versions of the MTG are likely to have a WiFi card as development continues. Additionally the future versions will run on the Linux operating environment which may serve to solve some of our issues with altering WiFi connections in app. See Issues and Challenges section for more on WiFi and potential solutions.

Software Specifications

The application is only supported for iOS 7 and up (current version is iOS 8). Additional frameworks for development may include Augmented Reality software for iOS and image recognition software.

Currently Vuforia is the primary option for AR software for iOS considering it is freely licensed to use and widely accepted and documented. See more information on Vuforia Developer Portal here: https://developer.vuforia.com/

Modelling

- An initial prototype displaying basic functionality and proof of concept to be completed by the end of the first semester.
- Additionally monthly demonstrations will show progress for the application throughout.
- Regular monthly check ins with ISG provided quick feedback to assist in our iterative development cycles.

Issues and Challenges

- Image Recognition on Mobile Devices
- Learning curve for developing with iOS, mobile development, and Augmented Reality
- Security limitations of the iOS system which prevent easy handling of WiFi and other built in libraries provided by the Apple system.

Testing Procedures and Specifications

Testing for application interfacing is manageable through the iOS simulator available with Apple's Xcode software. This software will allow our team to test and debug the fundamental usability of the application but will not be sufficient to fully test the Augmented Reality and connectivity of the application.

XCode provides Unit Testing capabilities as of version 5. However the way which it handles of the unit testing does not facilitate our applications process well. XCode unit testing doesn't allow for set up or tear down which does not play well with our applications database driven systems.

For testing we developed a simple server which could emulate an MTG connection over WiFi and constructed an auditing process which allows us to run batch processes and compare the database output to a defined 'correct' state.

An iPhone 4 has been provided by ISG for testing of the application which will serve to test the interface and hardware capabilities of the application. ISG will also provide a harness with an out of body MTG when testing reaches such a stage which requires testing on the physical hardware.

Further Resources and Documents

The human interface will interact in accordance with Apple's design guidelines and John Deere's trademark guidelines detailed in the links below. The interface must emphasize simplicity and ease of use to in order to meet the applications purpose and targeted user base.

The application will be interfaced with via the iPhone's capacitive screen and therefore must maintain a human interface easily operated by touch.

Minimum expected screen size is 3.5 (iPhone 4) to 5.5 (iPhone 6 Plus). Minimum resolution on the LCD screen is expected to be minimum 960×640 (iPhone 4) to 1920×1080 (iPhone 6).

For more information on designing for iOS see:

https://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/

Web resources for development

John Deere Developer Resources:

https://developer.deere.com/#/home/landing

Additional Guidance

Information on John Deere's trademark guidelines:

http://www.deere.com/en_US/docs/Corporate/citizenship/john_deere_tm_guides_sponsorships.pdf

Summary and Conclusion

This application is fully expected to be complete within the two allotted Semesters for this project and with careful selection of 3rd party software the project should come well within the $1000 budget

Appendix A Internal Module Interfaces

Internal Module Interfaces are Managers which assist the application with its basic data handling. Their job is primarily to handle interactions with the database as safely as possible.

Class references and descriptions of their functions can be found in the Documentation in both HTML and LaTeX format. Additional details on the function of Controllers and Views have been included as well for reference:

http://may1505.ece.iastate.edu/docs/html/index.html

Managers:

Database Manager

Class: `JDDatabaseManager`
Provides general access to the database. This class is generally used by the rest of the managers. Rarely should this class be used alone unless seeking access to application core settings information.

Also provides the field array class which is used to set and retrieve data sets.
Class: `JDFieldArray`
Responses from Database get requests are returned as NSArray *'s of these fields.
Technically the class extends NSArray * however the functionality is not used.
This class is used for both setting, getting and returning data. For example in order to set values in the database a new field array is created and the `addSetField` method is used and the field array is passed to the database's set method. When retrieving information from the database `addGetField` is used instead when passing to the database classes get method. The returned field array on a get will have all the same fields as the array passed in but associated values will be set and can be retrieved with the `getValueFor` method and the name of the field.

For learning how the JDFieldArray works it is recommended that future developers observe how data is access through it in the `JDDatabaseManager.m` file and in the various View Controllers that use them (`JDTractorInfoViewController.m`, e.g.).

Tractor Manager

Class: `JDTractorManager`
Provides getters and setters for vehicle information. See Class documentation for further method descriptions.

Maintenance Manager

Class: `JDMaintenanceManager`
Provides getters and setters for Maintenance information. See Class documentation for further method descriptions.

User Manager

Class: `JDUserManager`
Provides getters and setters for user information and is used to hash, salt passwords and check user's login attempts. See Class documentation for further method descriptions.

Download Manager

Class: `JDDownloadManager`
The Download Manager provides an interface to connect to the MTG and download maintenance information. It conforms to the protocol onboard the MTG for serving information to the application. The size of the incoming data structure is defined in the class header file and must be updated when the size of the structure *from the MTG* changes.

After initialization the manager can connect to a given IP and port. Using its download methods, various maintenance data information can be retrieved from the server on the MTG.

See Class documentation for further method descriptions.

Alert Manager

Class: `JDAlertManagerImproved`
The Alert Manager Improved class replaces the old obsolete Alert Manager. It provides methods for checking for alerts from the database. See the Class documentation for further specifics. This class behaves somewhat uniquely.

Helpers

Tractor List Helper

Class: `JDTractorList`
The tractor list helper (`JDTractorList`) reads the `tractors.xml` file (in `resources/`) and produces a list of tractors available to the application. All images for equipment in `tractors.xml` will need to be loaded into the bundle manually at this time. Currently

images are in `images/tractors/` on the file system. Their location in the bundle is under the same group structure: `images/tractors/`.

Threshold List Helper

Class: `JDThresholdList`
This helper provides a list of threshold parameters and other helpful maintenance data information. It is used to configure specifics on how to display maintenance information on a per tractor basis and is also used by the Alerts Manager to check alert status. Each tractor in the Tractor List Helper should have a corresponding .xml file entry in the Thresholds.

Threshold description .xml files are in the `images/tractors/` directory on the file system and a similar group path in the bundle. The name of each threshold xml file must start with the name of the tractor matching ***exactly*** the name specified in the `tractors.xml` file followed by `-thresholds.xml`. For example the entry for the 8R Tractor will have name `8R Tractor-thresholds.xml`.

Formatting of the .xml file looks like this:

```
<!-- Defines parameter thresholds for 8R Tractor -->
<root>
    <parameter>
        <name>Fuel Level</name>
        <test>below</test>
        <unit>gallons</unit>
        <type>int</type>
        <user>farmer</user>
        <threshold>65</threshold>
    </parameter>

    <parameter>
        <name>Engine Hours</name>
        <test>above</test>
        <unit>hours</unit>
        <type>int</type>
        <user>farmer</user>
        <threshold>50</threshold>
    </parameter>

    <parameter>
        <name>MTG Serial Number</name>
        <test>none</test>
        <unit> </unit>
        <type>string</type>
        <user>dealer</user>
        <threshold>0</threshold>
    </parameter>
    <parameter>
        <name>GPS Available</name>
        <test>below</test>
        <unit> </unit>
        <type>bool</type>
        <user>farmer</user>
        <threshold>1</threshold>
    </parameter>

    <!--Continue here… -->
</root>
```

The name field must match EXACTLY the name of the value as it is inserted into the database or the Helper cannot match the parameter.

Possible test values are above, below, and none. With above alerts are triggered when the stored value is above the threshold value, with below alerts are triggered when the stored value is below the threshold value.

Unit is the string value to call each unit of the threshold type (e.g. gallons, etc.). For best results values with no unit should have a single space.

Type is the parameter unit type. Pertinent values are int, bool, and string.

User indicates the level of user which this value is displayed for. Pertinent values are farmer, dealer, and developer.

Threshold is the value at which to trigger an alert. For Boolean use 0 and 1.

Appendix B Developer Manual

Appendix B Developer Manual is intended to provide a detailed and specific description of how the application has been implemented. The purpose is to facilitate future maintenance.

This appendix contains technical details involving the Apple iOS platform. Familiarity with the Apple Developer documentation and development platform is recommended.

This appendix provides usage information on A.R.T.I.M.U.S' internal workings and thus should not be shared unless with permission of John Deere ISG.

User Interface

The application user interface is defined primarily in the `Main.storyboard` file. The transitions between screens are, in most cases, defined by segues in this file except where more convenient to do so programmatically.

The various screens are described here:

Log In Navigation

The initial startup process involves a splash screen and prompts the user to log into their user space in the application.

Splash Screen

Display the John Deere logo prior to opening the application.

Login Screen

Log in as a current user.

Sign Up Screen

Add a new user to the phone.

Tabs Controller

This controls how tabs are switched between. This is handled mostly by Apple's own application interface. The setup is constructed in the `Main.storyboard`.

Tractor Vision Tab

This tab is for the AR Tractor Vision display information.

Vehicles Tab

This tab allows the user to display and edit Vehicle information.

Alerts Tab

Display global alerts for this user.

Settings Tab

Give access to options to adjust settings.

Tractor Vision View/Controller

Tractor Vision

My Vehicles View/Controller

My Vehicles handles interaction with specific vehicles. Screens in the tab are controlled by a Navigation Controller which roots in the Vehicles List. The navigation stack is defined almost entirely in `Main.storyboard`. Some segues are executed from code.

Vehicle List

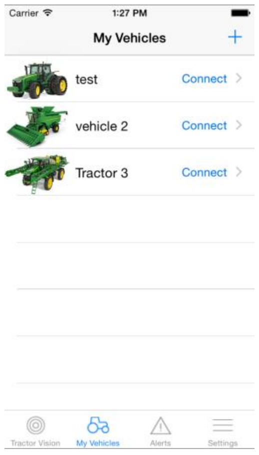


Figure 6: Vehicles List Screen

Class: `JDMyVehiclesTableViewController`
Extends: `UITableViewController`

The primary screen for the *My Vehicles* page is the list of vehicles. All vehicles listed on this page are from the current logged in user. This is the first page that is displayed when the *My Vehicles* tab is selected and is the root Controller for the Vehicles Navigation Controller.

Table cells are custom cells of class `JDVehicleTableCell`. The layout is defined in `JDVehicleTableCell.xib` with the file owner set to

`JDMyVehiclesTableViewController` (important for Connect Button).

Add Vehicle

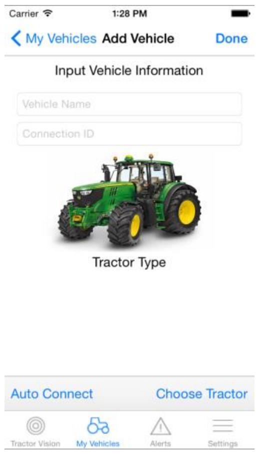
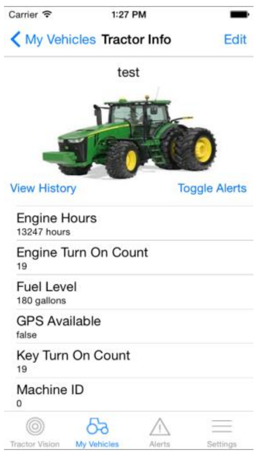


Figure 7: Add Vehicle Screen

Class: `JDAddVehicleViewController`
Extends: `UIViewController`

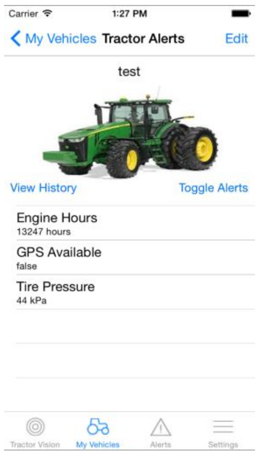
Add a vehicle to the current users list of vehicles accessed by clicking the "+" button. The layout is primarily defined in `Main.storyboard` with Equipment Selection Popup from the `JDTractorSelectPopupView.xib` layout. The equipment selection popup class is `JDTractorSelectPopupView`.

Vehicle Information



Engine Hours
13247 hours
Engine Turn On Count
19
Fuel Level
180 gallons
GPS Available
false
Key Turn On Count
19
Machine ID
0

Figure 9: Tractor Information Screen



Engine Hours
13247 hours
GPS Available
false
Tire Pressure
44 kPa

Figure 8: Tractor Alerts Screen

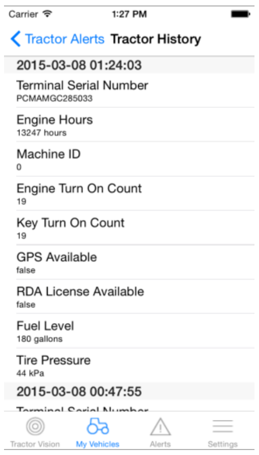
Class: `JDTractorInfoViewController`
Extends: `UITableViewController`

View information on the current tractor. There are two paths to this screen: if the user selects the connect button it will segue to this screen and attempt to download maintenance information on this tractor, if the tractor cell is selected it will simply move to the screen and not attempt an update. Both of these segues are executed from code.

The layout for this screen uses a custom cell for the first row of the table view. This cell has a layout from `JDTractorInfoHeaderCell` class and `.xib` file.

This screen allows the user to toggle between standard information and Alert information when a button from the `JDTractorInfoHeaderCell` is pressed. Additionally it can transition to the Vehicle History on a similar button push.

Vehicle History



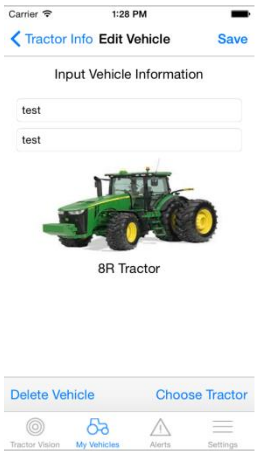
2015-03-08 01:24:03
Terminal Serial Number
PCMAMGC285033
Engine Hours
13247 hours
Machine ID
0
Engine Turn On Count
19
Key Turn On Count
19
GPS Available
false
RDA License Available
false
Fuel Level
180 gallons
Tire Pressure
44 kPa
2015-03-08 00:47:55
Terminal Serial Number

Figure 10: Vehicle History Screen

Class: `JDTractorHistoryTableViewController`
Extends: `UITableViewController`

Displays a list of all vehicle history. Maintenance responses are limited to a value defined in the `JDTractorHistoryTableViewController` header file, default is 300.

Edit Vehicle



test

test

Figure 11: Edit Vehicle Screen

Class: `JDEditVehicleViewController`
Extends: `UIViewController`

This screen is essentially a clone of the Add Vehicle page accessible by selecting the 'Edit' button from the *Vehicle Information* page. It will update a vehicle already in the database.

Delete Vehicle

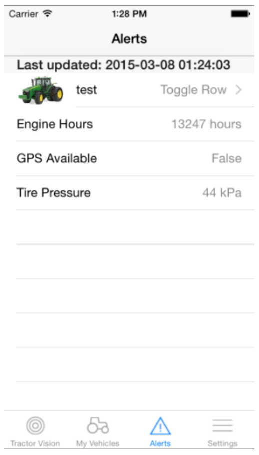
Class: `JDDeleteVehicleViewController`
Extends: `UIViewController`

This screen allows a vehicle to be deactivated, it is accessible by selecting the 'Delete' button from the *Edit Vehicle* page. It will turn a vehicle already in the database from *enabled* to *disabled*.

Alerts View/Controller

The Alerts page displays global alerts for a user. There is not a deeper navigation stack than the top page. The screen set up is defined almost entirely in `Main.storyboard`.

Alerts List



Last updated: 2015-03-08 01:24:03

test

Engine Hours	13247 hours
GPS Available	False
Tire Pressure	44 kPa

Figure 12: Global Alerts Screen

Class: `JDAlertsTableViewController`
Extends: `UITableViewController`

This page lists all the tractors that currently have alerts on them and allows the user to toggle open and closed the tractors alerts. If the user selects an alert it will add a check mark in the cell to help the owner indicated resolved issues until the next update.

Data Management

The application uses the native SQLite 3 API available to iOS in order to store and manage data. The Database Manager creates 2 databases on the device (locally): `AppCore.db` and `AppData.db`.

AppCore

`AppCore.db` is used for internal settings that are configurable by the user. It is also used to store the state (logged-in, not logged-in, e.g.) of a user and the current user ID. It does not store specific user information. All inputs stored in AppCore are strings.

To access AppCore simply initialize the `JDDatabaseManager` with the `init` method. This automatically connects to the `AppCore.db` database and accesses settings. In order to add or set a new AppCore setting entry use the `setAppCoreEntry` method, in order to get a value use `getAppCoreEntry`. Both of these methods are only useful if initialized with `init`. All values are set and retrieved as `NSString`s.

AppData

The `AppData.db` database is used for all dynamic data useful to the user. It is created by the application at first sign-up. All tables in the database are similarly created by the application as they are called by the application.

AppData contains information on users, tractors, maintenance information, etc. The respective managers are responsible for creating their own tables in the database. The Database Manager has methods to automate creation of tables and for adding, editing, and retrieving data from databases. Using the `JDDatabaseManager` with `initWithDatabaseFilename` and specifying `AppData` connects to the AppData database.

The `setEntries` and `getEntries` methods allow for adding/editing and retrieval of data from the database.

See Class documentation for `JDDatabaseManager` for further information on database management.

Database Structure

Listed here is the basic database structure for the AppData database:

```
// Fields for Maintenance table creation
NSArray *fieldsArray = [[NSArray alloc] initWithObjects:
                        @"VehicleID INTEGER",
                        @"FieldName TEXT",
                        @"FieldType INTEGER",
                        @"FieldTextValue TEXT",
                        @"FieldNumberValue REAL",
                        @"IsChecked INTEGER",
                        @"DateUpdated TIMESTAMP DEFAULT CURRENT_TIMESTAMP",
                        @"FOREIGN KEY(VehicleID) REFERENCES Vehicles(ID)",
                        nil];

// Fields for Users table creation
NSArray *fieldsArray = [[NSArray alloc] initWithObjects:
                        @"UserFirstName TEXT",
                        @"UserLastName TEXT",
                        @"UserName TEXT",
                        @"UserEmail TEXT",
                        @"UserPasswordHash TEXT",
                        @"UserSaltValue TEXT",
                        @"UserType INTEGER",
                        @"IsEnabled INTEGER",
                        nil];

// Fields for Vehicles table creation
NSArray *fieldsArray = [[NSArray alloc] initWithObjects:
                        @"VehicleName TEXT",
                        @"VehicleType TEXT",
                        @"VehicleImage TEXT",
                        @"ConnectionName TEXT",
                        @"UserID INTEGER",
                        @"IsEnabled INTEGER",
                        @"FOREIGN KEY(UserID) REFERENCES Users(ID)",
                        nil];
```

This information is pertinent for use with the `JDFieldArray` class when using the `getValueFor:` method.

WiFi Connectivity

WiFi connectivity through this app is not supported as Apple has deprecated all access to the WiFi or settings.