# 



Kinetic Mobile – Sprint 1

Contents

[Loading Image 3](#_Toc480276424)

App Branding 3

Login Screen 3

Home Screen 3

Dashboard 3

Menu 3

Historical Navigation 3

Log Out 3

Authentication 3

Initial Sync 3

Error Logging 3

Sync on Login 3

Data Access 3

Terminology Changer 3

Job Schedule 3

Job List Screen 3

Settings 3

Help & Support 3

# Loading Image

Add the initial application loading screen which will display for 3 seconds on application launch. The background and kinetic image will be in the images folder and should have the following layout.



Figure Background



Figure 2 Loading Image Desktop and Mobile

The Kinetic logo should fade from left to right whilst displayed.

# App Branding

A company will be able to add their company logo to the top left-hand of the application and the login screen.

After successfully pinging the SyncService, do a “GET” request to the “Branding” service. This can be accessed at <http://.../SyncService.svc/Branding>.

This will return a JSON response with the following model:

    public class AppBrandingResponse

    {

        public byte[] LoginLogo { get; set; }

        public byte[] AppLogo { get; set; }

    }

The logos should be saved for future in the “ApplicationInfo” table. After setting the service, this should be set at application launch.

In order to provide client specific customisation of the application in the future, styles such as fonts and colours should be pulled from a model to be used in the frontend. No colours should be hardcoded in the frontend.

# Service Screen

When the application launches for the first time, they will be required to input the service that they want to connect to. They will not need to input the fully qualified URL, just the service which will be placed into the format logic is below:

The service box should look like this:

[                  ].motionkinetic.net

(i.e. have .motionkinetic.net after it)

The text box will support these formats of url:

1. Appv2 – just the instance name, this is what all live customers will use
2. Kinetic.motioniis/appv2 – this is what we use for internal testing
3. localhost:1234/SyncService/SyncService.svc – used for local development
4. motionws00/SyncService/SyncService.svc - used for local development

The logic would be as follows. “text” is what is entered into the textbox, everything is case insensitive:

If text ends with “syncservice”

            url = “http://”+text+”/SyncService.svc”

else if text ends with “.motionkinetic.net”

            remove “.motionkinetic.net” from the text

            url = “http://”+text+”syncservice.motionkinetic.net/SyncService.svc”

else if text contains “.” Or “/” or “localhost” or “motionws”

            url = “http://”+text+”/SyncService/SyncService.svc”

else

            url = “http://”+text+”syncservice.motionkinetic.net/SyncService.svc”

The application will then ping the URL to ensure that the service is valid using the “ping” (no parameters) method of the web service. If the service cannot be foun­­­d, an error message should be displayed to the user saying the following:

“The service cannot be found. Please ensure you have the correct service and that your device has an internet connection”

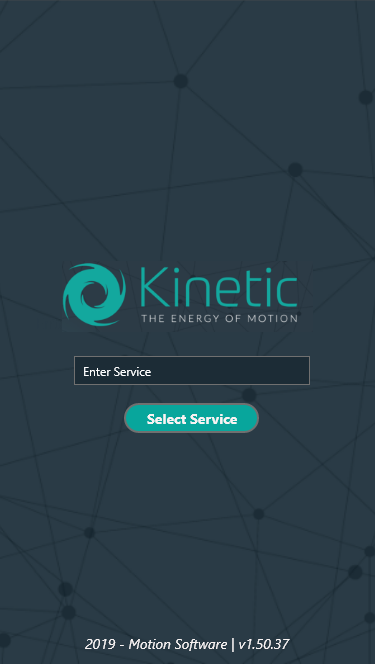
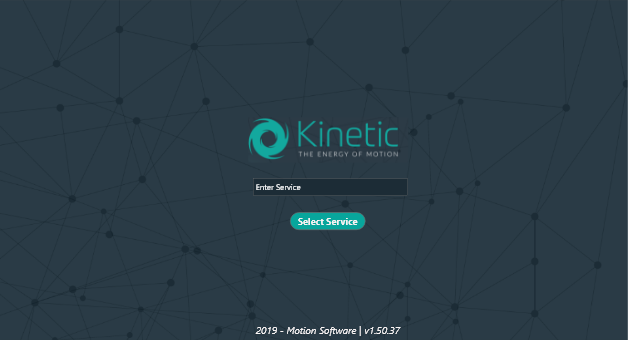
If the service is found, the URL will be stored in the “Service” column of the “ApplicationInfo” table, which is in the StructureDatabase. The company logo should be pulled now from the service via the Branding method (see App Branding) on successful verification of the service.

Figure 3 Service Screen Desktop and Mobile

The Service screen will look like figure 2.

If the credentials have not be authenticated by Kinetic, the user will see “The Service you have entered is incorrect”. The ping method signature is:

[WebGet]

[OperationContract]

bool Ping();

The application should receive a message of true for verification. If timed out, display the “The Service you have entered is incorrect” message.

Within the top section of the panel there should be a placeholder for Kinetic branding to go in. This is pulled down from the web service and then inserted into the “ApplicationInfo” table.

Below this, the Enter Service field should be contained within a rectangular box a #1C2C36 background.

Enter Service Field:

Placeholder – “Enter Service” positioned to the left within the box. Once the user begins to type in this box, the text “Enter Service” should appear above the box, aligned on the left. The text should be #808080

A line of white text should be centred along the bottom of the screen to indicate the app version number. The format should be as follows:

[Year] *–* Motion Software | v [version number]

# Login Screen

Depending on whether the device user exists in the database, the user will either go straight to the Dashboard screen or if they don’t exist in the database, they will be required to be authenticated by Kinetic.

## New Users

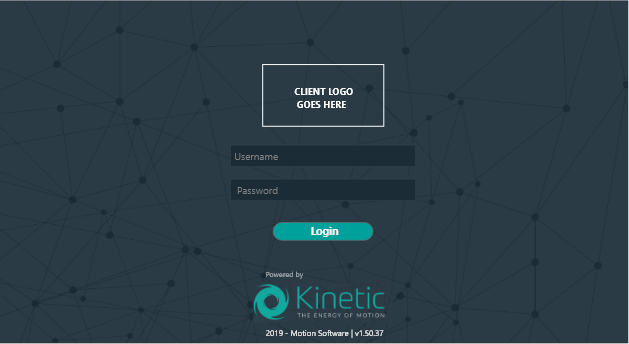


Figure 2 Login Screen Landscape/Portrait

The login screen will look like figure 3. A new user will be required to input their Kinetic username and password which is sent to the Kinetic web service for authentication. If the credentials have not be authenticated by Kinetic, the user will see “The Username/Password you have entered is incorrect”. If successful, the service will send an access token which is stored in the local database.

A rectangular panel should be centred on the screen which will contain all login and branding information.

Within the top section of the panel there should be a placeholder for the client branding to go in. This is pulled down from the web service and then inserted into the “ApplicationInfo” table.

If there are not logos found in the database, use the default Motion Software brand logo.

Within the main body of the panel, the Login fields should be contained with a white background.

Username field:

Placeholder – “Username”

Password field:

Placeholder – “Password”

Format – Hidden text displayed as dots

Login button:

Text – “Login”

Style – Fit the full white login panel. The button background colour will be #00a79d with white text.

Within the bottom section of the panel, there should be a placeholder for the Kinetic branding to go in. This is pulled down from the web service and then inserted into the “ApplicationInfo” table.

A line of white text should be centred along the bottom of the screen to indicate the app version number. The format should be as follows:

©[Year] *–* Motion Software | v [version number]

The application will send an authenticat ion request to the following WCF method:

[OperationContract]

Task<AuthenticationResponse> Authenticate(AuthenticationRequest request);

AuthenticationRequest contains the following parameters:

public class AuthenticationRequest

{

public string Username { get; set; }

public string Password { get; set; }

}

If the credentials have not be authenticated by Kinetic, the user will see “The Service you have entered is incorrect”. The application will receive an authentication response:

public bool IsAuthenticated { get; set; }

public bool IsServiceAvailable { get; set; } = true;

public string Message { get; set; }

public double? ServerUTCDate { get; set; }

public Guid? UserID { get; set; }

public Guid? AccessToken { get; set; }

public Guid? PersonID { get; set; }

If successful, the app will provide the user with an access token for further communication with the server and take the user to the Login screen. The user name, access token and PersonID should be stored in the user table. Not all user will have a PersonID. There should only ever be one entry in the user table. If one exists before, truncate and then add the new user.

## Existing Users

If the user is found in the local database in the user table, the user will not be required to enter their Kinetic credentials. The user will be loaded and the application will use the access token for all communication with the web service.

Supplementary details can be found in the Mobile Server Interaction document.

# Home Screen

The home screen will be displayed to the user after successfully logging into the system and provides the initial navigation routes.

The main screen will be split into two sections: home and dashboard which is covered in a separate task.

Use the Telerik tabbed view to create the main home page with the “Home” and “Dashboard” options. The tabs will be clickable, and a swipe gesture should be added to allow the user to swipe between the tabs. The current selected tab should have a highlighted in #00a79d bar beneath the text with unselected tabs white. The tab text will be #2c3b45.

## Home Tab

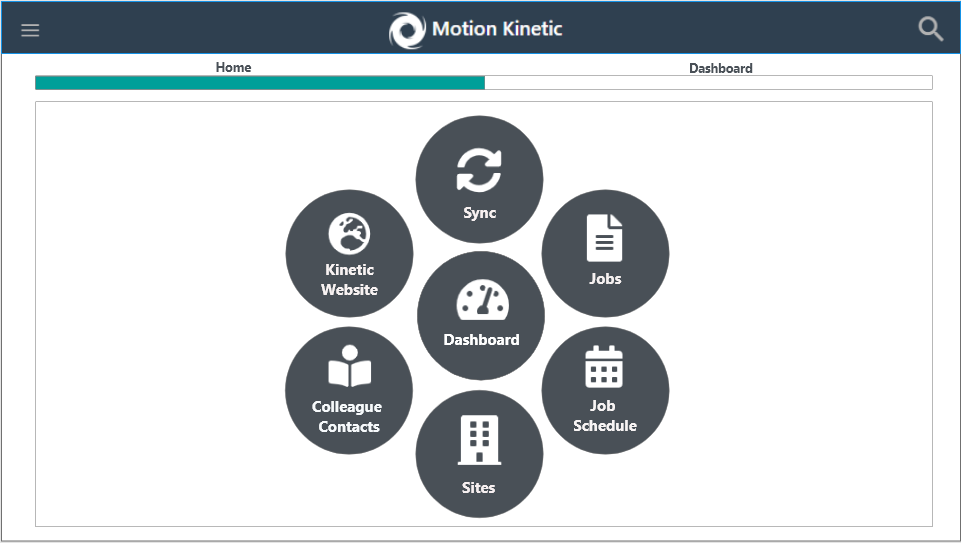
The Home tab will be made up of seven circles, each of which will navigate the user to a different section of the system.



Figure 3 Navigation as circles – Portrait/Landscape

The circles will have a #2c3b45 background with the Icon at the top and centred text beneath, both in white.

The home page will have the following options (clockwise):

1. Jobs

Icon: file-alt

Text: Jobs

Style: #2c3b45 background in the circle; white text; white icon

Navigates to: Job list



1. Sites

Icon: building

Text: Sites

Style: #2c3b45 background in the circle; white text; white icon

Navigates to: Sites

1. Colleague Contacts

Icon: book-reader

Text: Colleague Contacts

Style: #2c3b45 background in the circle; white text; white icon

Navigates to: Contacts

1. Website

Icon: global-europe

Text: Kinetic Website

Style: #2c3b45 background in the circle; white text; white icon

Navigates to: Opens browser to their instance of the main Kinetic site

1. Job Schedule

Icon: calendar-alt

Text: Job Schedule

Style: #2c3b45 background in the circle; white text; white icon

Navigates to: Job Schedule



1. Run sync

Icon: sync-alt

Text: Sync

Style: #2c3b45 background in the circle; white text; white icon

Navigates to: Sync Screen



1. Dashboard

Icon: tachometer-alt

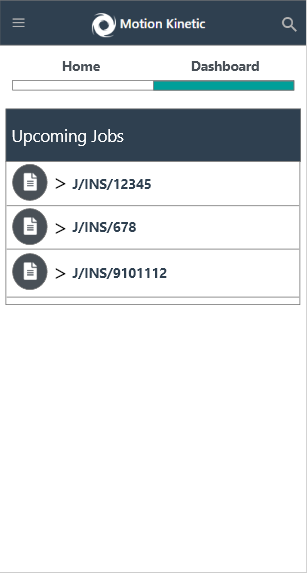
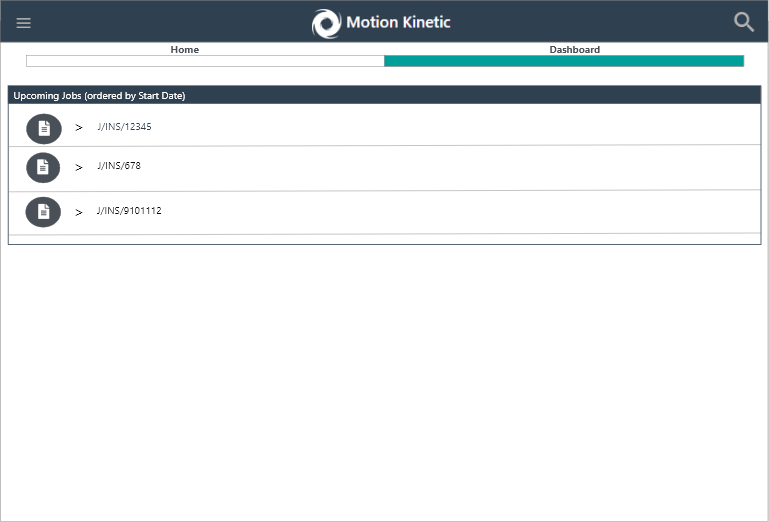
Text: “Dashboard”

Style: #2c3b45 background in the circle; white text; white icon

Navigates to: Dashboard tab

# Dashboard

The dashboard panel will display three lists displayed in Telerik Expander controls (<https://www.telerik.com/xamarin-ui/expander>). The main expander bar will have a #2c3b45 background with white text. A white chevron expander icon will sit in the far-right hand side on the bar.

The panel will have the file-alt icon on a #2c3b45 circular background with white text. To the right of the icon will be the Job number. Clicking on this row will navigate to the Job Details screen.

Panel: Upcoming Jobs (ordered by Start Date)

Overview – Display the next three jobs allocated to the user by Start Date

# Menu

Add a burger bar button to the top left hand side of the main panel. Use the Telerik side drawer (<https://www.telerik.com/xamarin-ui/sidedrawer>) to add the side menu options.

The top of the menu should have the customer brand logo. Underneath this will be the menu options that are available for the user to click on.

When the menu is displayed, the remainder of the screen should be blurred as to indicate that the menu is engaged and only the details on the menu are relevant at that time. This should happen “out of the box” with the side drawer control.

Clicking anywhere off the menu, on the screen, will collapse the menu and return the user to the screen that is behind the menu.

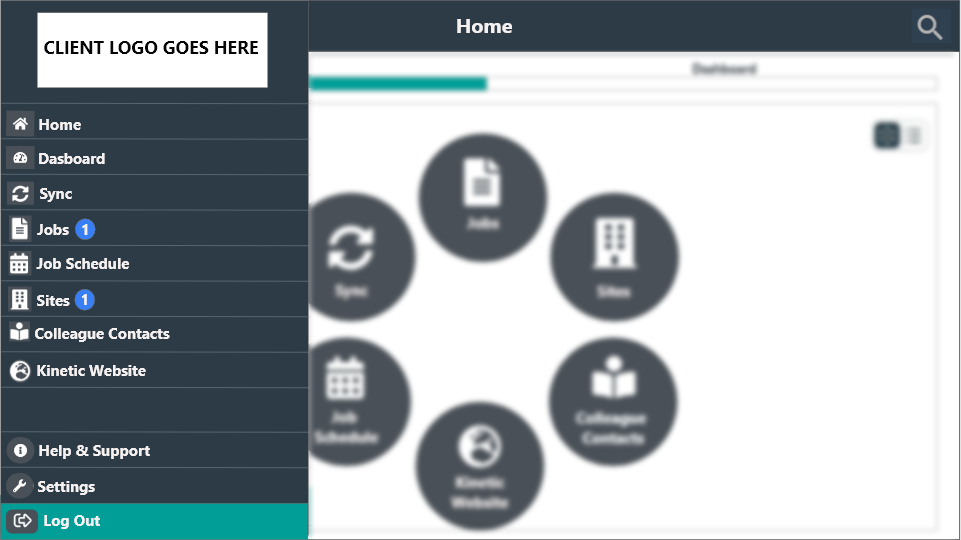


Figure 4 Side drawer menu

The menu options will be displayed in the following order:

1. Home

Icon: home

Text: “Home”

Style: #2c3b45 background spanning entire panel; white text; white icon

Navigates to: Home with the Home tab selected

1. Dashboard

Icon: tachometer-alt

Text: “Dashboard”

Style: #2c3b45 background spanning entire panel; white text; white icon

Navigates to: Home screen with Dashboard tab selected.

1. Sync

Icon: sync-alt

Text: “Sync”

Style: #2c3b45 background spanning entire panel; white text; white icon

Navigates to: Sync screen

1. Job

Icon: file-alt

Text: “Jobs”

Style: #2c3b45 background spanning entire panel; white text; white icon

Navigates to: Job list screen

1. Schedule

Icon: calendar-alt

Text: “Job Schedule”

Style: #2c3b45 background spanning entire panel; white text; white icon

Navigates to: Job Schedule screen

1. Sites

Icon: building

Text: “Sites”

Style: #2c3b45 background spanning entire panel; white text; white icon

Navigates to: Sites screen

1. Contacts

Icon: book-reader

Text: “Colleague Contacts”

Style: #2c3b45 background spanning entire panel; white text; white icon

Navigates to: Contacts screen



1. Kinetic Website

Icon: global-Europe

Text: “Kinetic Website”

Style: #2c3b45 background spanning entire panel; white text; white icon

Navigates to: Opens the default browser to the company’s Kinetic instance.



1. Help and support

Icon: info-circle

Text: “Help & Support”

Style: #2c3b45 background spanning entire panel; white text; white icon

Navigates to: Help and support screen

1. Settings

Icon: wrench

Text: “Settings”

Style: #2c3b45 background spanning entire panel; white text; white icon

Navigates to: Settings screen

1. Log out button

Icon: unlock

Text: “Log out”

Text Style: #08A69C background spanning entire panel; white text; white icon

Navigates to: Login screen. Clears current session objects.

# Historical Navigation

## Breadcrumbs

Add a generic method to add and remove breadcrumbs to the main page as not all pages will require being added to the breadcrumb panel with the page to navigate too, breadcrumb text and page parameters. On press or click of the breadcrumb link, the user should navigate to the applicable page.

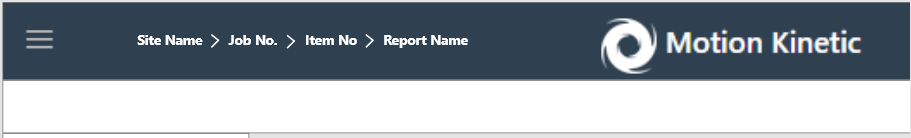
The breadcrumbs should be contained within the screen header bar, to the left of the screen title and just to the right of the hamburger button.

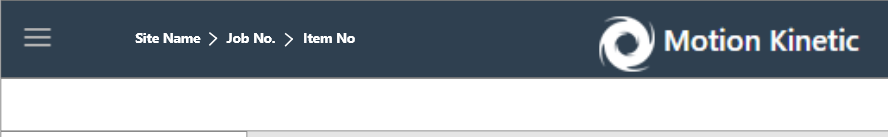
The breadcrumbs represent the hierarchy of Kinetic (Site-Job-Item-Report). These should always show the latest of the following places that have been accessed:

* Site
* Job
* Item
* Report

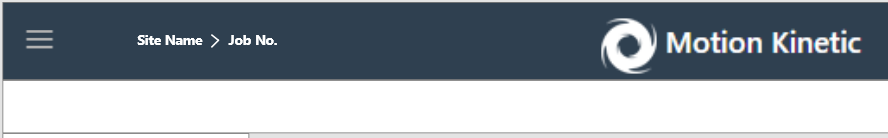
1. When the user is on levels 2 (Job), 3 (Item) or 4 (Report) the previous levels should be displayed and should display the name/no,

e.g.

When on Level 4 (Report), the breadcrumbs should be displayed as follows,

When on Level 3 (Item) the breadcrumbs should be displayed as follows,

When on Level 2 (Job) the breadcrumbs should be displayed as follows,



When on Level 1 (Site) the breadcrumbs should be displayed as follows,



1. The navigation should work as follows,

* When the user selects the Site, they should be navigated to that specific Site
* When the user selects the Job No., they should be navigated to that specific Job’s Job Details Screen
* When the user selects the Item No., they should be navigated to that specific Item Tab screen,
* When the user selects the Report No., they should be navigated to that specific Report’s View Screen

[For the mobile app, we need to consider how the breadcrumbs will work on a smaller screen. One suggestion is to have a dropdown in the header bar with the levels presented vertically from top level (site) to bottom level (report). Another is to strip the breadcrumbs out of the header and have a separate breadcrumb bar just below the header bar.]

## History Dropdown

As the user is in the Kinetic Mobile Application, the last 10 pages accessed by the user is to be stored. Use a singleton service for this so that it can be stored in memory.

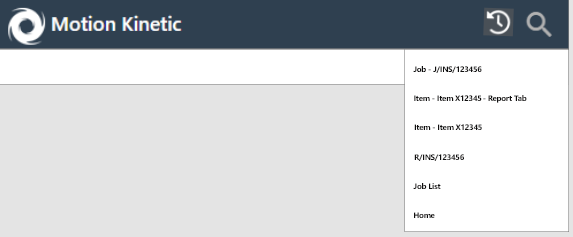
Each user should be able to access their last 10 pages using a button that will be positioned at the left-hand side of the breadcrumb. Either use a Xamarin Forms image button or use this link to create one using the Font Awesome font <https://forums.xamarin.com/discussion/comment/350244/#Comment_350244>.

The button should be a history icon. The icon colour should be white with a #2c3b45 background



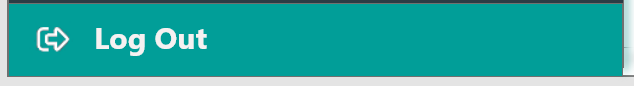
Add a helper method to generate the text on the history list following these rules:

* + - When a Report Screen - **Report – [Insert Report No]**
    - When an Item Screen - **Item [Insert Item No]**
    - When a Job Screen - **Job – [Insert Job No]**
    - When a Job Form Screen - **Job Form – [Insert Form Name]**
    - When a Site Screen - **Site – [Insert Site Name]**
    - Default - e.g. – **Dashboard, Home, Job Schedule**

The history list will appear as a dropdown below the button on large screens but as a Xamarin Forms popup on small screens. Most recent screen at the top of the list. If you click on a history item, the history item moves to the top and doesn’t duplicate.

# Log Out

The logout button will be located at the bottom of the side menu. When the user clicks this, the applicable user row in the user table will be removed, the back stack will be reset and the user will be navigated to the Login screen.



Call the Kinetic service method:

[OperationContract]

Task<bool> RemoveAccessToken(RequestBase request);

With parameters:

[DataContract]

public class RequestBase

{

[DataMember]

public string Username { get; set; }

[DataMember]

public string Password { get; set; }

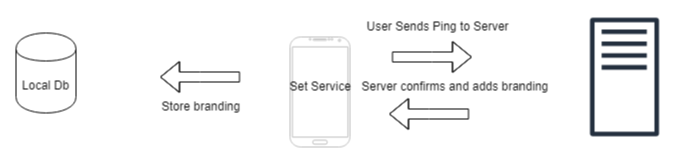
}

Passing in the applicable users access token as the password to remove the token entry from the main Kinetic system.

# Authentication

Authentication will be carried out in two ways depending on whether there is an entry in the user table. If there is an entry, this user should be used for authentication. There should only be one user in this table at any one time.

## Login for New Users



When the user initial starts the application, they will be presented with the service input screen. On this screen, the user will specify the service name that they want to connect to. They will not be required to type the entire URL as the application will construct this for them, i.e. https://www.{service name}.com/. The device will then make a call to check whether this URL is valid. If the URL is valid, the server will respond with the branding information to the application which will be stored in the structure database. The application will then navigate to the user login screen (with the branding displayed) for user validation.



The login screen will operate like a standard login screen. The user will input their username and password to be authenticated by the main Kinetic system. If the credentials supplied by the user are valid, the server will respond with a hashed and salted user access token which will allow the user to interact with the main Kinetic system. This token will be stored in the User database with the username of the logged in device user and the Kinetic user that was used for authentication.

## Login for Existing User



If there is a row in the user session database, the user will be able to gain access to the application without inputting their credentials. The application should open at the home screen using the username and the authentication token from the user row.

# Initial Sync

The initial sync will run the same type of sync as the when a user logs into the application but the screen required will be different.

A user will be able to either perform a data upload, a data download or a full sync.

For all of the steps below, use the SyncServiceURL and append the method name. E.g. if the sync service url is <http://appv2syncservice.motionkinetic.net/SyncService.svc> then the service call would be <http://appv2syncservice.motionkinetic.net/SyncService.svc/GetApplicationInfo>. All methods are POST unless otherwise stated. All content is sent in JSON format.

### Step 1. Call “GetApplicationInfo”

**Request**

Include username and access token in standard format:

{"username":"john.smith","password":"e116427f-8c3d-ad8d-42e9-ae02e42fea95"}

**Response**

This will return results with the following model:

|  |
| --- |
| public class ApplicationInfoModel  {  public string CustomerCode { get; set; }  public string CustomerName { get; set; }  public string ApplicationName { get; set; }  public bool HideScheduler { get; set; }  public string VersionNo { get; set; }  } |

This information should be stored in the App database

### Step 2. Call “PreSync”

**Request**

Include username and access token in standard format:

{"username":"john.smith","password":"e116427f-8c3d-ad8d-42e9-ae02e42fea95"}

**Response**

This will simply return a bool of “true”. No action is taken from this response.

### Step 3. Call “RequestServerSync”

**Request**

Include username and access token in standard format. Also include “type” and set to “initial”

{"username":"john.smith","password":"e116427f-8c3d-ad8d-42e9-ae02e42fea95","type":"initial"}

**Response**

This will return results with the following model:

|  |
| --- |
| public class ServerSyncDetails  {  public Guid SyncID { get; set; }  public int AuditBatchSize { get; set; }  } |

SyncID is used in future requests during the sync process. It links together the requests on the server side.

AuditBatchSize is used in the next step.

### Step 4. Call “UploadClientSyncAuditBatch”

**Request**

This sends to the server entries from the SyncAudit table. Include username and access token in standard format.

|  |
| --- |
| {  "clientAudits": [{  "ID": "8ad5ee13-05a5-4b79-8a9a-0ec6f8b34abe",  "Ticks": 1565268287042,  "IsDelete": 0  }, {  "ID": "3f16a2b9-a0ef-43dd-a5fa-1b887a5cca66",  "Ticks": 1565268287042,  "IsDelete": 0  }, {  "ID": "6066e1f2-e28c-4453-9f3c-2592a042fd90",  "Ticks": 1565268287042,  "IsDelete": 0  }  ],  "syncID": "aa933d0f-e20a-4c96-a9db-70f9e8e38d75",  "username": "motion",  "password": "e116427f-8c3d-ad8d-42e9-ae02e42fea95"  } |

ClientAudits is a list containing 3 columns from the SyncAudit table.

SyncID is taken from the response of “RequestServerSync” above.

As this is the “initial” sync, the query on the SyncAudit table to produce this data should be filtered down to the following “TableName”s (this list will grow during the project so ensure that it is easily modified).

* Address
* Contact
* Customer
* Job
* JobPerson
* LanguageText
* Setting
* PersonSchedulerEvent
* SchedulerEvent
* SchedulerEventItem
* Site
* UsergroupUser
* UsergroupPermission

**Response**

The server will return a bool: true for success, false for error. If an error occurred don’t proceed further with the sync and show the user a message.

### Step 5. Call “FinalizeClientSyncAuditUploadsV2”

**Request**

Include username and access token in standard format. Also include “type” set to “initial” and the SyncID from the response of “RequestServerSync” above.

{"username":"john.smith","password":"e116427f-8c3d-ad8d-42e9-ae02e42fea95","type":"initial", “SyncID”:” 690dbf6b-4df1-fb82-447c-5a3fd072de55”}

**Response**

|  |
| --- |
| public class ServerDataSyncResponse  {  public int BatchCount { get; set;}  public Guid SyncID { get; set; }  } |

BatchCount is how many times the next step is to be called.

SyncID is just a confirmation of the SyncID passed in and can be ignored.

### Step 6. Call “DataBatchV2” (loop)

This service will be called in a loop, the number of times it is looped will be the same as the BatchCount in the previous response. Include username and access token.

**Request**

|  |
| --- |
| public class ServerDataBatchRequest  {  public Guid SyncID { get; set; }  public int BatchNo { get; set; }  } |

SyncID is taken from the response of “RequestServerSync” above.

BatchNo is the current batch we are requesting, starting at 1.

**Response**

JSON Representation:

|  |
| --- |
| {  "ServerRows": [{  "ID": "8ad5ee13-05a5-4b79-8a9a-0ec6f8b34abe",  "Ticks": 1565268287042,  "Type": “Job”,  “Model”: “{\”SomeJSONString\”:123456}”  }, {  "ID": "8ad5ee13-05a5-4b79-8a9a-0ec6f8b34abe",  "Ticks": 1565268287042,  "Type": “Job”,  “Model”: “{\”SomeJSONString\”:123456}”  }, {  "ID": "8ad5ee13-05a5-4b79-8a9a-0ec6f8b34abe",  "Ticks": 1565268287042,  "Type": “Job”,  “Model”: “{\”SomeJSONString\”:123456}”  }  ],  "BatchNo": "1",  } |

Where a Filestore type is returned, also note this ID in memory as it is used in the next step.

Class representation:

|  |
| --- |
| public class ServerDataBatchV2Response  {  public List<ServerRowV2Response> ServerRows { get; set; }  public int BatchNo { get; set; }  }  public class ServerRowV2Response  {  public Guid ID { get; set; }  public string Model { get; set; }  public long Ticks { get; set; }  public string Type { get; set; }  } |

### Step 7. Call “FilestoreV2” (loop)

For each FilestoreID returned from DataBatchV2, we need to download the actual data. For each Filestore object returned in DataBatchV2, download the actual bytes using this method.

**Request**

|  |
| --- |
| public class FilestoreChunkRequest  {  public string SyncID { get; set; }  public string FileID { get; set;}  } |

SyncID is taken from the response of “RequestServerSync” above.

FileID is taken from the DataBatchV2 response above.

**Response**

This will return the raw bytes, it will not be wrapped in a JSON call. Files should be saved in the file system with the filename as the GUID and no file extension.

### Step 7. Call “CalculateDefinitionDownloads”

**Request**

Include username and access token in standard format. Also include the SyncID from the response of “RequestServerSync” above.

{"username":"john.smith","password":"e116427f-8c3d-ad8d-42e9-ae02e42fea95", “SyncID”:” 690dbf6b-4df1-fb82-447c-5a3fd072de55”}

**Response**

|  |
| --- |
| public class CalculateDefinitionsResponse  {  public int BatchCount { get; set; }  public Guid SyncID { get; set; }  } |

BatchCount is how many times the next step is to be called.

SyncID is just a confirmation of the SyncID passed in and can be ignored.

## Step 8. Call “GetDefinitionBatchV2” (loop)

**Request**

|  |
| --- |
| public class GetDefinitionBatchReqeust  {  public Guid SyncID { get; set; }  public int BatchNo { get; set; }  } |

SyncID is taken from the response of “RequestServerSync” above.

BatchNo is the current batch we are requesting, starting at 1.

**Response**

|  |
| --- |
| {  "ServerRows": [{  "SyncAuditTicks": 1565268287042,  "Type": “Job”,  “Object”: “{\”SomeJSONString\”:123456}”  }, {  "ID": "8ad5ee13-05a5-4b79-8a9a-0ec6f8b34abe",  "SyncAuditTicks": 1565268287042,  "Type": “Item”,  “Object”: “{\”SomeJSONString\”:123456}”  }, {  "ID": "8ad5ee13-05a5-4b79-8a9a-0ec6f8b34abe",  "SyncAuditTicks": 1565268287042,  "Type": “Report”,  “Object”: “{\”SomeJSONString\”:123456}”  }  ],  "BatchNo": "1",  } |

|  |
| --- |
| public class DefinitionBatchV2Response  {  public List<DefinitionRowV2Response> ServerRows { get; set; }  public int BatchNo { get; set; }  }  public class DefinitionRowV2Response  {  public Guid? ID { get; set; }  public string Object { get; set; }  public long SyncAuditTicks { get; set; }  public string Type { get; set; }  } |

These rows will be saved into the “Definition” table.

### Step 9. Call “UploadClientDataModelAudits”

**Request**

This sends to the server entries from the “Data” table. Include username and access token in standard format.

|  |
| --- |
| {  "dataModelAudits": [{  "ID": "8ad5ee13-05a5-4b79-8a9a-0ec6f8b34abe",  "Ticks": 1565268287042,  "Type": “Item”  }, {  "ID": "3f16a2b9-a0ef-43dd-a5fa-1b887a5cca66",  "Ticks": 1565268287042,  "Type": “Item”  }, {  "ID": "6066e1f2-e28c-4453-9f3c-2592a042fd90",  "Ticks": 1565268287042,  "Type": “Report”  }  ],  "syncID": "aa933d0f-e20a-4c96-a9db-70f9e8e38d75",  "username": "motion",  "password": "e116427f-8c3d-ad8d-42e9-ae02e42fea95"  } |

dataModelAudits is a list containing 3 relevant columns table.

SyncID is taken from the response of “RequestServerSync” above.

As this is the “initial” sync, the query on the “Data” table to produce this data should be filtered down to the following “TableName”s:

* Site
* Job
* Customer

**Response**

The server will return a bool: true for success, false for error. If an error occurred don’t proceed further with the sync and show the user a message.

### Step 10. Call “CalculateDataModelDownload”

**Request**

Include username and access token in standard format. Also “type” set to “initial” and include the SyncID from the response of “RequestServerSync” above.

{"username":"john.smith","password":"e116427f-8c3d-ad8d-42e9-ae02e42fea95", type":"initial",“SyncID”:” 690dbf6b-4df1-fb82-447c-5a3fd072de55”}

**Response**

|  |
| --- |
| public class CalculateDataModelResponse  {  public int BatchCount { get; set; }  public Guid SyncID { get; set; }  } |

BatchCount is how many times the next step is to be called.

SyncID is just a confirmation of the SyncID passed in and can be ignored.

## Step 11. Call “GetDataModelBatchV2” (loop)

**Request**

|  |
| --- |
| public class GetDefinitionBatchReqeust  {  public Guid SyncID { get; set; }  public int BatchNo { get; set; }  } |

SyncID is taken from the response of “RequestServerSync” above.

BatchNo is the current batch we are requesting, starting at 1.

**Response**

|  |
| --- |
| {  "ServerRows": [{  "ID": "8ad5ee13-05a5-4b79-8a9a-0ec6f8b34abe",  "SyncAuditTicks": 1565268287042,  "Type": “Item”,  “Object”: “{\”SomeJSONString\”:123456}”  }, {  "ID": "8ad5ee13-05a5-4b79-8a9a-0ec6f8b34abe",  "SyncAuditTicks": 1565268287042,  "Type": “Report”,  “Object”: “{\”SomeJSONString\”:123456}”  }  ],  "BatchNo": "1",  } |

|  |
| --- |
| public class DataModelBatchV2Response  {  public List<AppDataModel> ServerRows { get; set; }  public int BatchNo { get; set; }  }  public class AppDataModel  {  public Guid ID { get; set; }  public string Type { get; set; }  public string Object { get; set; }  public long SyncAuditTicks { get; set; }  } |

These rows will be saved into the “Data” table.

# Error Logging

All errors should be recorded. If a user has an internet connection, this should send error details automatically, otherwise they should be stored locally and sent as part of the upload process. Once the error messages are sent, the table should be truncated. The messages should be stored as the JSON string that would have been sent over to the Kinetic log service.

If the user is online, the server will respond with an error code. This should be presented to the user in a popup.

Code snippet shown below from current app:

Model

|  |
| --- |
| public string Platform { get; set; } //Always “App”         public string Environment { get; set; } //Server side enum, can be “Test”/”Live”/”Motion”. See below for calculation          public DateTime CreatedDate { get; set; }          public string Customer { get; set; } //CustomerName          public string Username { get; set; } //Logged in user’s name          public string UserID { get; set; } //Logged in user’s ID          public string URL { get; set; } //Not applicable to the app, leave blank          public string Hostname { get; set; } //KineticWebsite URL (as opposed to the sync service, see previous working out of this)          public string VersionNo { get; set; } //Version No of the app         public string ExceptionType { get; set; }          public string Message { get; set; }          public string StackTrace { get; set; }          public string InnerExceptionType { get; set; }          public string InnerMessage { get; set; }          public string InnerStackTrace { get; set; }          public dynamic AdditionalData { get; set; } //JSON object that can be used to pass additional data to the error logging server, leave blank when not sure |

The way to calculate the environment:

If url contains “test” then

            Environment = “Test”

If url does NOT contain “motionkinetic.net” but DOES contain “motion”

            Environment = “Motion”

If url contains “localhost”

            Environment = “Motion”

If url contains “motionkinetic.net” (but not “test”)

            Environment = “Live”

Application code:

|  |
| --- |
| var json = JsonConvert.SerializeObject(errorModel, new JsonSerializerSettings() { NullValueHandling = NullValueHandling.Ignore });              using (var webClient = new WebClient())              {                  webClient.Headers.Add(HttpRequestHeader.ContentType, "text/json");                  var result = webClient.UploadString("<https://log.motionkinetic.net/api/log>", json);                  dynamic data = JsonConvert.DeserializeObject(result);                  if (data.ShortNo != null)                  {                      return data.ShortNo;                  }                  return null;              } |

# Sync on Login

When the user first logs into the system, the base information is downloaded from the Kinetic Service. This includes the header level job information.

The data will be sent in a number of batches in order to combat memory issues on certain mobile devices and will be in a JSON format with the object type at the top of the tree and a JSONified array of objects following e.g.

“item” :

{

“item1”:

{

},

“item2”:

{

},

…

}

The idea is to have a timer-based control to display the current percentage complete for the download. The percentage should be based on the current batch number compared with the total number of batches to download, rounded to the nearest whole number.

We would like the loader to be generic so that it can be used in large mode (below) and for in rows – small mode. It should also be possible to change the message or hide all the text completely.

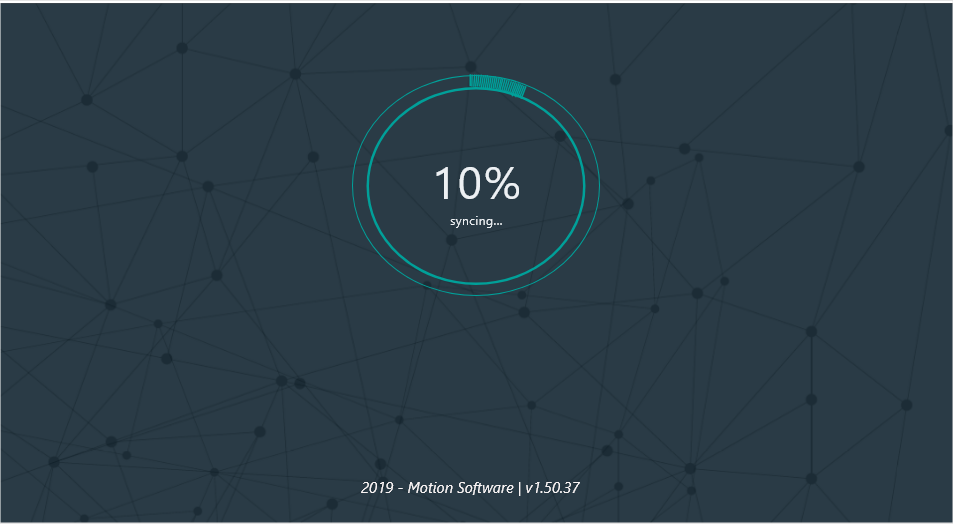


Figure 5 Concept for progress control

When the download has finished the control will be fully filled in with a colour of #00a79d and the text within will read “100%” with “Sync complete” written underneath. This message should be displayed for one second then navigate the user to the Home screen with the Dashboard tab selected.

This sync pulls in the initial data required for the user to use the systems such as user details, jobs etc… The sync process is described in a separate document.

Once the batches have been communicated to the device, it will loop round and insert the data batch by batch. The service will send data models back to be insert into the data definitions and the data table.

This sync will have to be accessible by other screens and split into an upload sync and a download sync.

# Data Access

The main point of data storage will be a SQLite database and should be accessed using Entity Framework. There will be a Data Model project within the project solution containing the POCO classes which should be used to build the database using a code first methodology.

When the user first launches the app, the database should be created and all data migrations (created using the code first process) ready for data to be transferred using the data syncs. All subsequent logins must ensure that the data migrations have all be executed on app launch.

The data model project will be shared with the main Kinetic main system to ensure that the data that is sent from the main system adheres to the same data structure.

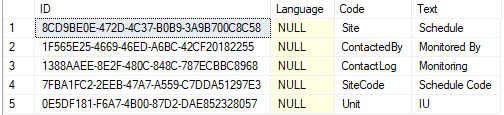
In order to follow the “separation of concerns” principle, the read and write actions of the database should be split into separate namespaces and folders, but remain in the same project.

The system will also have two databases, the core structure database (containing app information, error logs, etc… Appendix 1) and the data store database (containing the actual working data such as data models, data definitions, etc… Appendix 3).

# Terminology Changer

Clients will be able to change the terminology in various sections of the system. We will have default terminology where the user has not added custom terminology. There will be a code which will be used to access the terminology from the Language Text database table. The terminology will be sync’d as part of the initial sync. Ideally this would be accessed from memory to prevent any lag from querying the database.

The language table will look like the following:



Setup the mechanism which will be used by the various front ends for setting the screen terminology.

# Job Schedule

The UI will be split into two sections.

The main section (Desktop: to the left; Mobile: on top) will be a calendar view showing scheduled jobs for that current user which will be linked using a Xamarin calendar control. The user will be able to select a view mode (week/month/list where applicable) from a selector. Default the view mode to week.

The other section (Desktop: to the right; Mobile: on the bottom) will be a list view of the user’s unscheduled jobs. These will be listed as a white file-alt icon with a circular background #2c3b45 style to the left-hand side of the list followed by the job number in #2c3b45 text.

Both sections will be read-only and linked directly to the job details screen for the selected job.

Each entry on the calendar and unscheduled jobs should have:

Job Name,

Site Name,

Customer Name

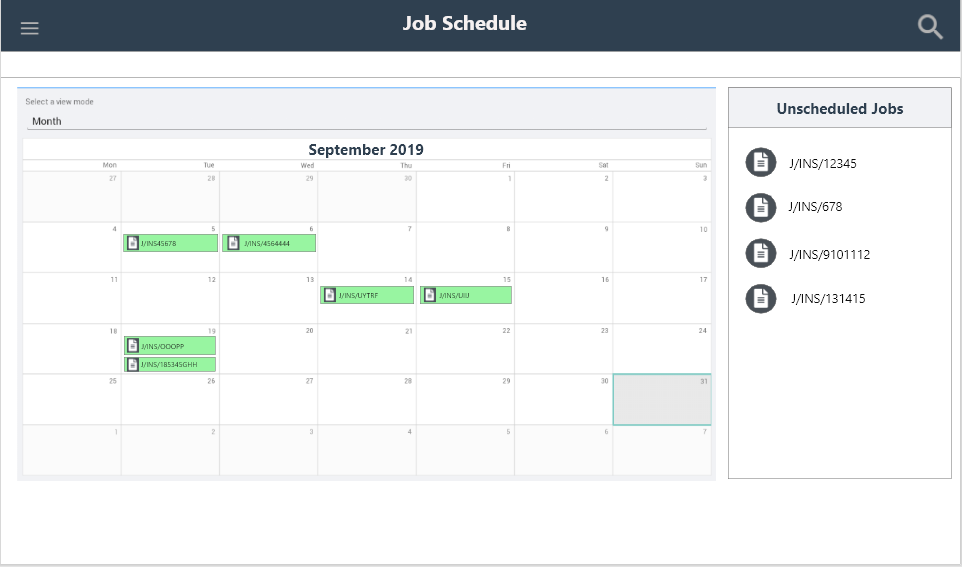


Figure Job Schedule

Unscheduled jobs will not have a SchedulerEventItem entry.

The schedules will come from the SchedulerEvent table linked to SchedulerEventItems. The user is linked to the SchedulerEvent by the PersonSchedulerEvent table.

If a schedulerEvent doesn’t have a schedulerEventItem, the start date is the EstimatedStartDate on the Job table. If this does not have an EstimatedStartDate user the StartDate.

The end date follows the same principle:

If a schedulerEvent doesn’t have a schedulerEventItem, the end date is the EstimatedEndDate on the Job table. If this does not have an EstimatedEndDate use the EndDate. If there’s no end date, use the startdate at 23:59.

To get the data start at the SchedulerEventItem > schedulerEvent > personschedulerevent > Job.

# Job List Screen

The user can access the Job List screen from the Main Menu panel by clicking on ‘Job List’.

This screen should display a list of all the available jobs assigned to the user, either in a downloaded or undownloaded state.

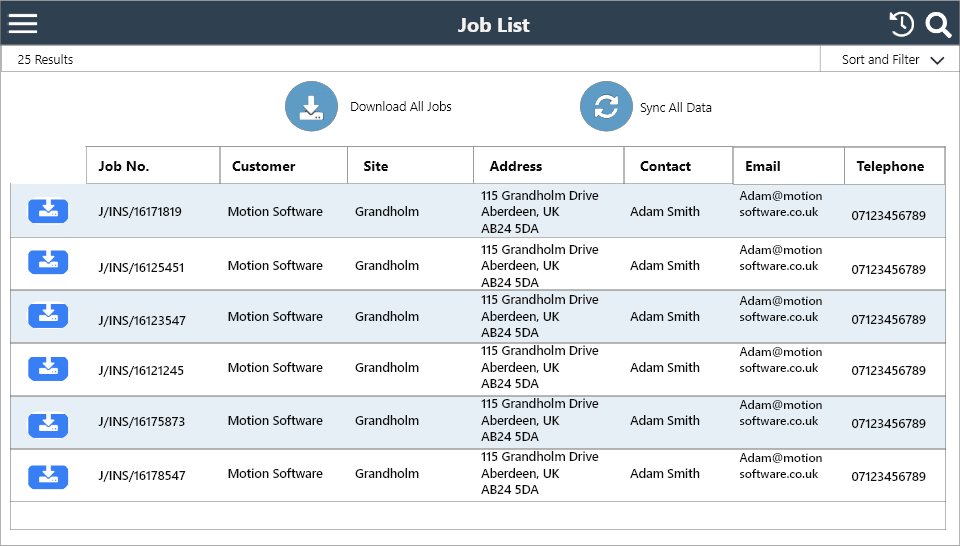
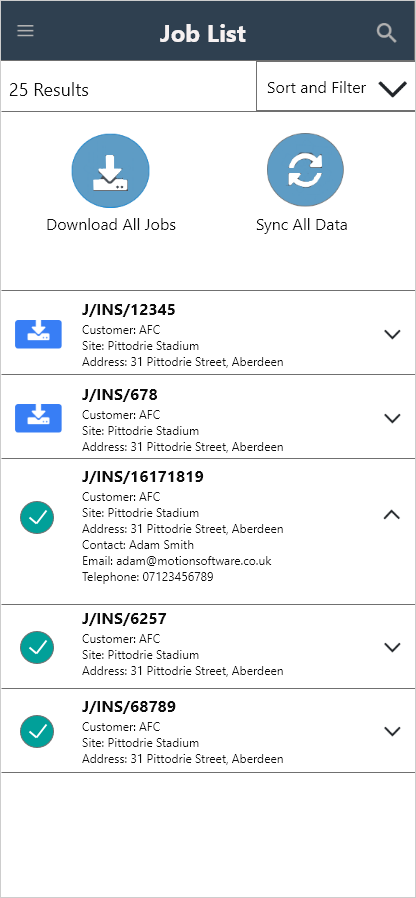
The data for the Job List can be found in the Job table of the Data Database and by default ordered by Start Date.

For larger screens, the results should be displayed in a grid. For smaller screens, the results should be shown in a card view.

There should be a download button (for jobs not yet downloaded) or a #00a79d circle with a white tick (for jobs that have been downloaded). If the download has failed, then a red circle with a white cross should be displayed instead.

The grid view should have alternative row shading and be clickable through to the Job Details screen for that job.

There The grid columns should be as follows:

* Job No
* Customer
* Site
* Address
* Contact
* Contact Email
* Contact Telephone

The card view should contain all of data for the columns in a stacked list underneath a dropdown represented by a chevron. Only the Job number (in bold), the customer, the site and the address should be visible. There should be a chevron expandable button on the right side of the card. When the user clicks on this button, the contact, email and telephone details will be presented.

There should be a download button (for jobs not yet downloaded) or a #00a79d circle with a white tick (for jobs that have been downloaded). If the download has failed, then a red circle with a white cross should be displayed instead.

Clicking anywhere else on the card should take the user to the Job Details screen for that job.

In the #2c3b45 screen header there should be a white heading, “Job List”, centred in middle of the screen. There should be a white burger button to the far left, for accessing the main menu, and there should be a white history button and a white search button to the far right. The history button will be a dropdown with the 10 most recent screens. The search button will be a global search of the system.

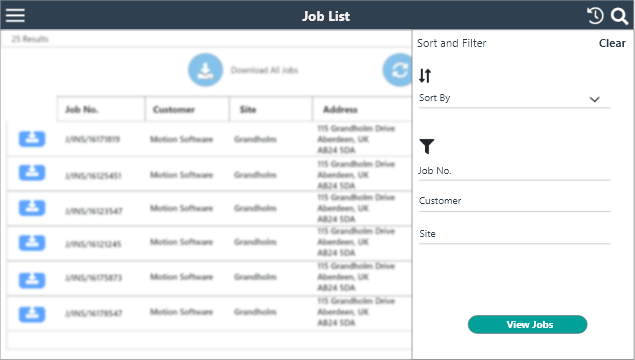
Directly under the screen header, a white bar with a grey border should exist across the entire width of the screen. Within this, the total number of jobs returned on the screen is displayed on the left side. The number is displayed followed by the text “Results” (e.g. 25 Results). A dropdown should be within this bar at the right side which will allow the user to sort and filter the results. This control should be called “Sort and Filter” and be side drawer Xamarin control. A #2c3b45 chevron should be placed to the right of the text.

Sort and Filter

When a sort and filter option has been applied, the Sort and Filter button on these screens should have a #2c3b45 tick to the left of the text. This will be removed once the user clears the sort and filter or leaves the Job List screen completely.

When the user clicks on this control the following happens:

a) Desktop



<https://www.youtube.com/watch?v=itfCvwZqR9g>

A side drawer panel over the screen at the right hand-side should appear and present the options to sort and filter the results. When the sort and filter panel is displayed, the remainder of the screen should be blurred as to indicate that the panel is engaged and only the details on the panel are relevant at that time.

Clicking anywhere off the panel, on the screen, will collapse the panel and return the user to the screen that is behind the menu. Any selections made by the user will not be applied in this case.

There should only be one option for the user to sort on:

1. Sort By – User should be able to click on the downward #2c3b45 chevron to present a list of options to sort by. These options should be:

* Start Date
* Job No
* Customer
* Site

Include a sort-alt icon at the top of this section. The icon should be #2c3b45 in colour.

There should be three options for the user to filter on:

1. Job No. – User should be able to type a job number in here. The field should be set to ‘contains’ in order to pick up any of the text entered and filter on that once ‘View Jobs’ button has been pressed.

2. Customer – User should be able to type in a customer name in here. The field should be set to ‘contains’ in order to pick up any of the text entered and filter on that once ‘View Jobs’ button has been pressed.

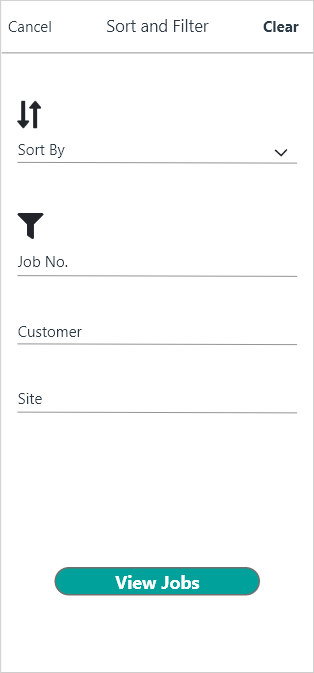
3. Site – User should be able to type in a site name in here. The field should be set to ‘contains’ in order to pick up any of the text entered and filter on that once ‘View Jobs’ button has been pressed.

Include a filter icon at the top of this section. The icon should be #2c3b45 in colour.

Perform sort and filter: click ‘View Jobs’ button after they have made their selections. Clicking this will refine the results in the list. For fields Job No., Customer and Site, all text in these should be considered when filtering and sorted as set.

Perform clear sort and filter: a clear button will be present in the top right-hand corner of the panel.

Other ways to clear the sort and filter will be to exit the Job List screen or to apply new sort and/or filters as this should clear the previous search and apply the new selections.

b) Mobile

When clicking on the Sort and Filter dropdown, a Sort and Filter screen should appear and fill the entire screen.

There should only be one option for the user to sort on:

1. Sort By – User should be able to click on the downward #2c3b45 chevron to present a list of options to sort by. These options should be:

* Start Date
* Job No
* Customer
* Site

Include a sort-alt icon at the top of this section. The icon should be #2c3b45 in colour.

There should be three options for the user to filter on:

1. Job No. – User should be able to type a job number in here. The field should be set to ‘contains’ in order to pick up any of the text entered and filter on that once ‘View Jobs’ button has been pressed.

2. Customer – User should be able to type in a customer name in here. The field should be set to ‘contains’ in order to pick up any of the text entered and filter on that once ‘View Jobs’ button has been pressed.

3. Site – User should be able to type in a site name in here. The field should be set to ‘contains’ in order to pick up any of the text entered and filter on that once ‘View Jobs’ button has been pressed.

Include a filter icon at the top of this section. The icon should be #2c3b45 in colour.

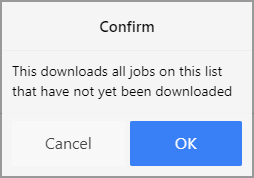
Perform sort and filter: click ‘View Jobs’ button after they have made their selections. Clicking this will refine the results in the list. For fields Job No., Customer and Site, all text in these should be considered when filtering and sorted as set.

Perform clear sort and filter: a clear button will be present in the top right-hand corner of the panel.

Other ways to clear the sort and filter will be to exit the Job List screen or to apply new sort and/or filters as this should clear the previous search and apply the new selections.

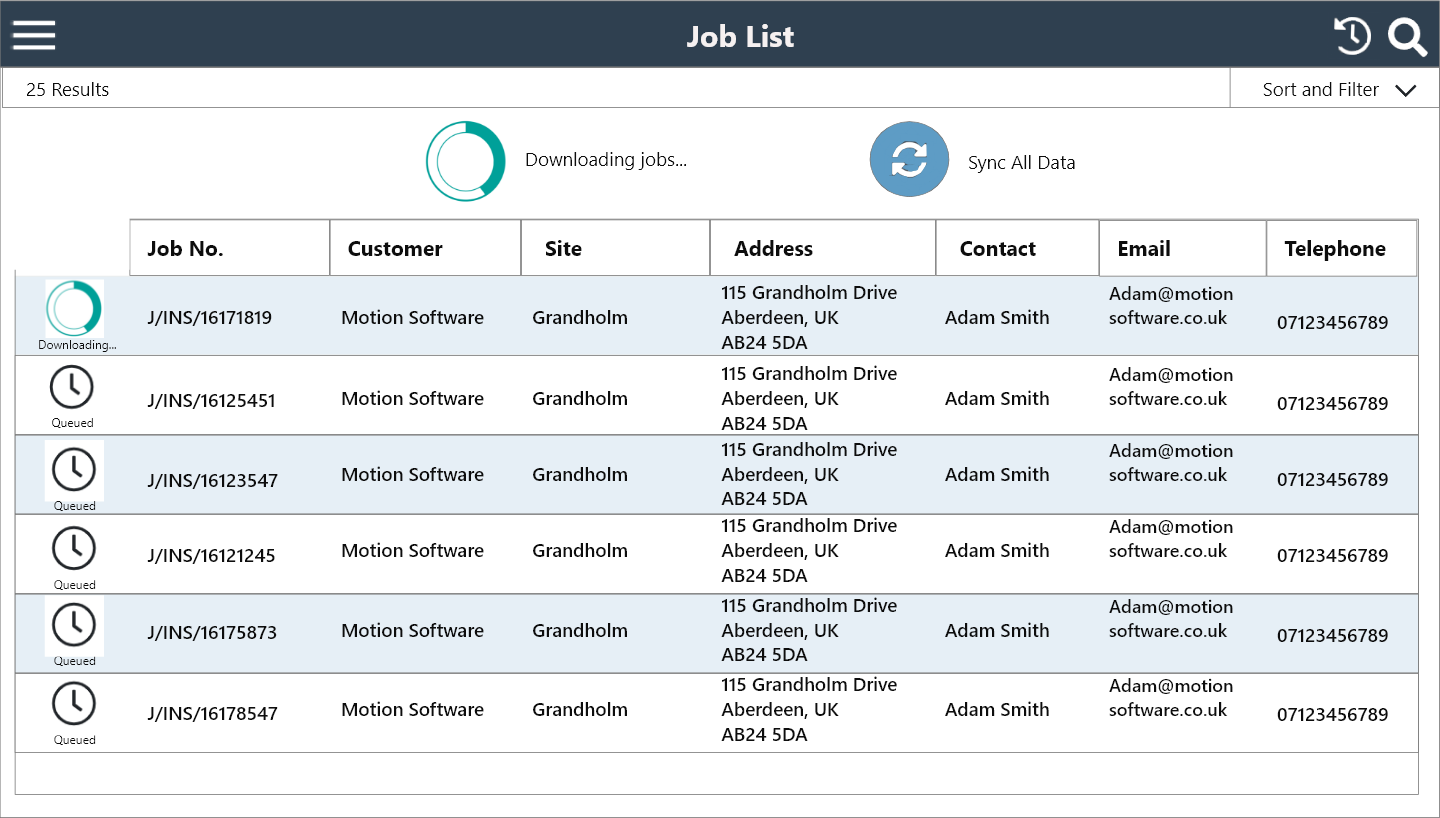
Download All Jobs

When the user clicks on the circular button, “Download all Jobs”, a Xamarin popup should appear.

This popup should give the user the choice between cancelling the action or performing with it.

a) Cancel – If the user clicks “Cancel” then the action is not performed, and the user is taken back to the Job List screen

b) OK – If the user clicks “OK” then the action is performed, and the user is taken back to the Job List and the system will go off and download all the jobs that are currently displayed on this screen.



When the system starts to download the jobs, starting with the job at the top of the list, the system will go down the list and download one at a time.

The “Download All Jobs” button will be removed and replaced with a timer-based control to display the current percentage complete for all job downloads. The percentage should be based on how far along in the total download process the system is (i.e. 10% of the way through all downloads, 10% of the control will be filled in with a colour of #00a79d). The text next to the button should now read “Downloading jobs…” and be in colour #2c3b45. There should be text in the middle of the button in #2c3b45 to display the percentage (e.g. “10%”).

The idea is for the download buttons down the left to be removed and replaced with the same timer-based control but no percentage to be displayed in the centre. The button should still be filled in to reflect how far along the download process is for the job (i.e. 10% of the way through, 10% of the control will be filled in with a colour of #00a79d). Below the button, the text should read “Downloading…”

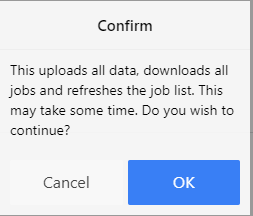
When the download has finished the control will disappear and be replaced with a filled in circle of colour #00a79d with a white tick in the centre. This indicates that the job has been fully downloaded.

Jobs that are waiting to be downloaded whilst another job is downloading are represented by a clock icon with text below which should read “Queued”. Both the icon and text should be in the colour #2c3b45.

Sync All Data



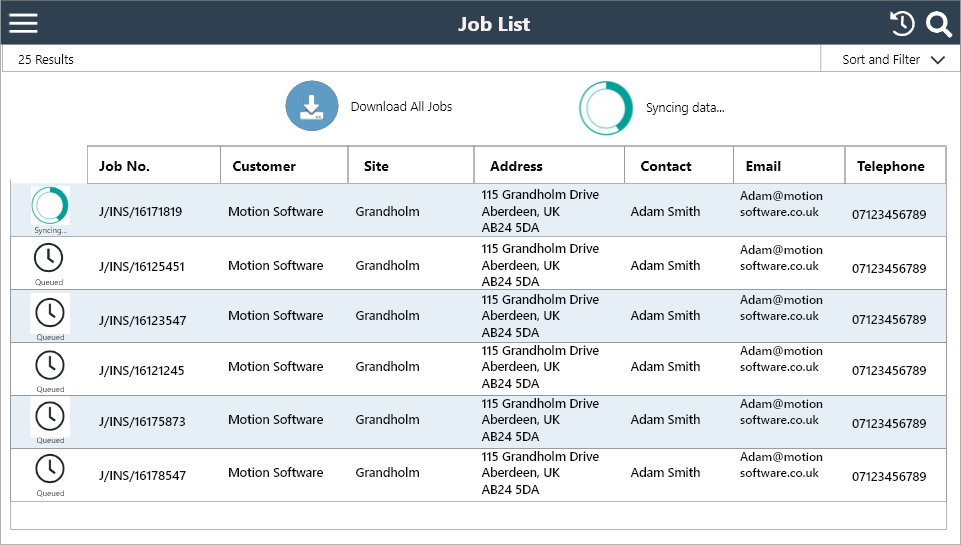
When the user clicks on the circular button, “Sync All Data”, a Xamarin popup should appear.

This popup should give the user the choice between cancelling the action or performing with it.

a) Cancel – If the user clicks “Cancel” then the action is not performed, and the user is taken back to the Job List screen

b) OK – If the user clicks “OK” then the action is performed, and the user is taken back to the Job List and the system will go off and upload all data; download all the jobs that are currently displayed on this screen; and refresh the job screen

When the system starts to sync the jobs, starting with the job at the top of the list, the system will go down the list and sync one at a time.



The “Sync All Data” button will be removed and replaced with a timer-based control to display the current percentage complete for all job syncs. The percentage should be based on how far along in the total sync process the system is (i.e. 10% of the way through all job syncs, 10% of the control will be filled in with a colour of #00a79d). The text next to the button should now read “Syncing jobs…” and be in colour #2c3b45. There should be text in the middle of the button in #2c3b45 to display the percentage (e.g. “10%”). The idea is for the download buttons down the left to be removed and replaced with the same timer-based control but no percentage to be displayed in the centre. The button should still be filled in to reflect how far along the download process is for the job (i.e. 10% of the way through, 10% of the control will be filled in with a colour of #00a79d). Below the button, the text should read “Syncing…”

We would like the loader to be generic so that it can be used in large mode and for in rows – small mode. It should also be possible to change the message or hide all the text completely.

When the sync has finished the control will disappear and be replaced with a filled in circle of colour #00a79d with a white tick in the centre. This indicates that the job has been fully downloaded.

Jobs that are waiting to be synced whilst another job is syncing are represented by a clock icon with text below which should read “Queued”. Both the icon and text should be in the colour #2c3b45.

**-- NOTE: Ensure that icons have transparent backgrounds**

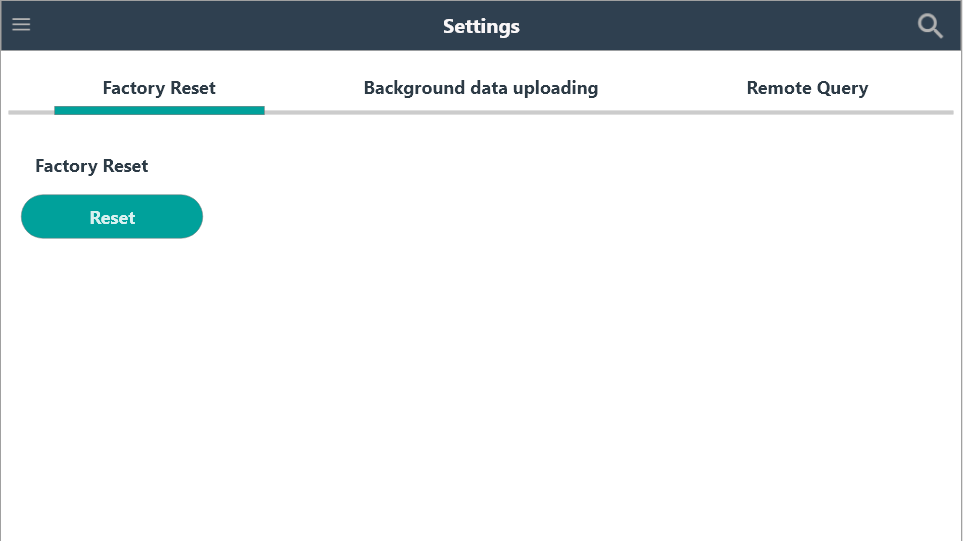
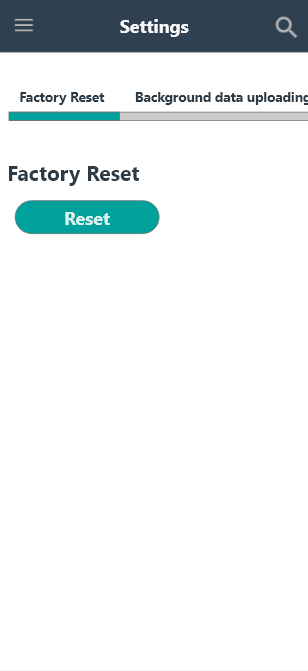
# Settings

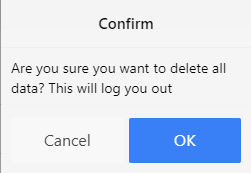
The Settings screen will be displayed to the user after clicking on the settings option in the main menu.

The screen will be split into three sections using the Telerik tabbed view.

The sections will be: Factory Reset; Background Data Uploading; and Remote Query.

The tabs will be clickable, and a swipe gesture should be added to allow the user to swipe between the tabs. The current selected tab should have a highlighted in #00a79d bar beneath the text with unselected tabs white. The tab text will be #2c3b45.

­­­Factory Reset

The user only has the option to perform a system factory reset by clicking on the ‘Reset’ button. If the user clicks on this, a Xamarin popup will appear to ask for confirmation.

This popup should give the user the choice between cancelling the action or performing with it.

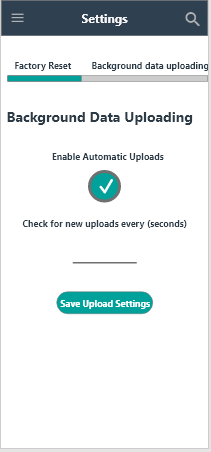
a) Cancel – If the user clicks “Cancel” then the action is not performed, and the user is taken back to the Factory Reset tab in the Settings screen.

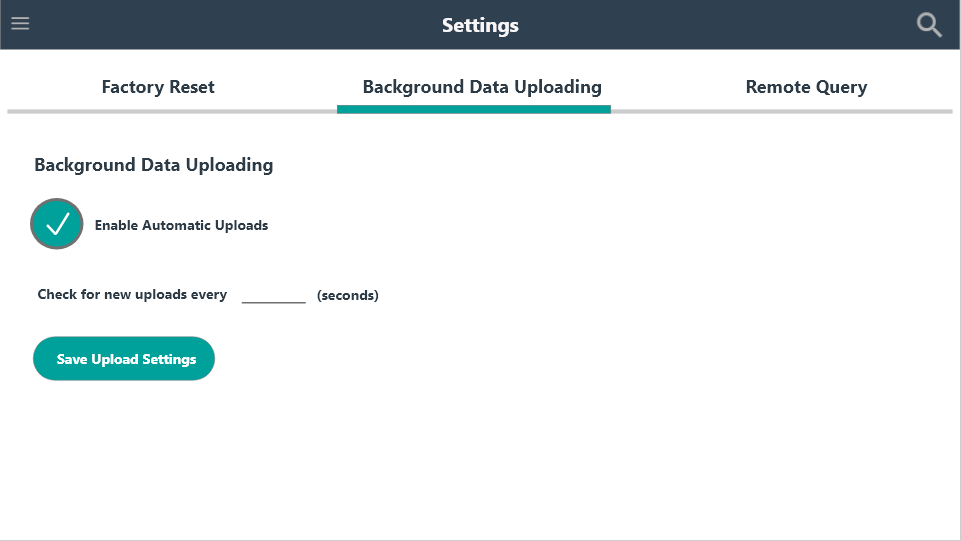
b) OK – If the user clicks “OK” then the action is performed, and the system deletes all data and performs the reset. The logout process as defined in the logout task is executed.

Background Data Uploading

Add a new tab to the settings screen called “Background Data Uploading”.

The background data uploading tab will allow the user to apply a frequency, in seconds, that the system will automatically upload data back to the main system. The minimum frequency that can be set should be 30 seconds, if enabled.





When the user clicks the save button, a Xamarin popup message should appear to indicate that the setting has been saved.

Remote Query

Add a new tab to the settings screen called “Remote Query”.

The remote query tab will allow data fixes to be run on the database, but a query should only execute on the chosen database. For the query, add a text area and a button to fire the query.

There should be two options for choose from: AppDB or DataDB.

The user inputs a code into the text box which corresponds to a script on the server. The script is then downloaded and then runs on the selected Database. The reason for this function is to provide Motion with a way of executing data fixes remotely. A code is sent to a centralised server away from the sync service. This then returns a query string to be executed. The URL and code snippet is below.

The signature for this method is parameterless from this URL:

<http://app.motioninspect.com/InspectCentralAppServer/Service1.svc/GetQuery>

<http://app.motioninspect.com/InspectCentralAppServer/Service1.svc/ReturnQueryResults>

factory.processCode = function(code, databaseName) {

var request = {

Code: code

};

return qhttp.post(getQueryUrl, request)

.then(function(queryResult) {

return processQuery(queryResult, databaseName, code);

});

}

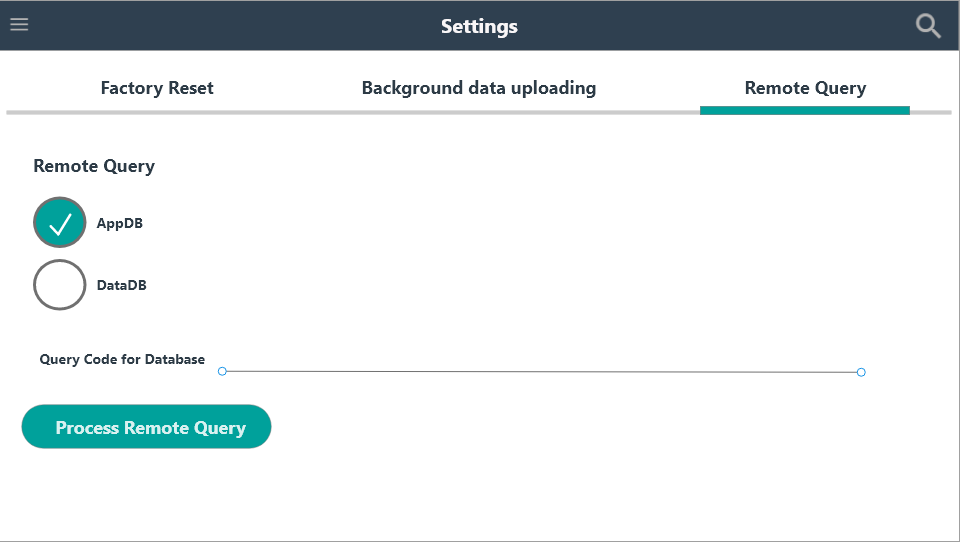
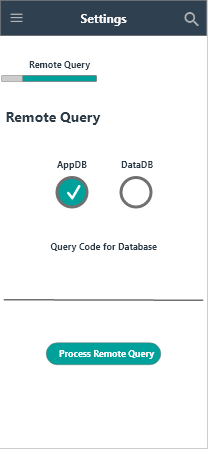
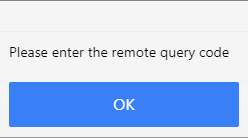
 

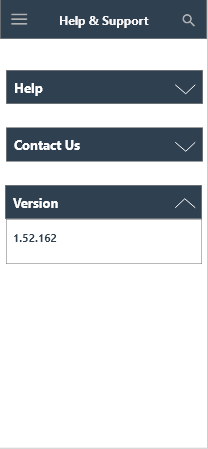
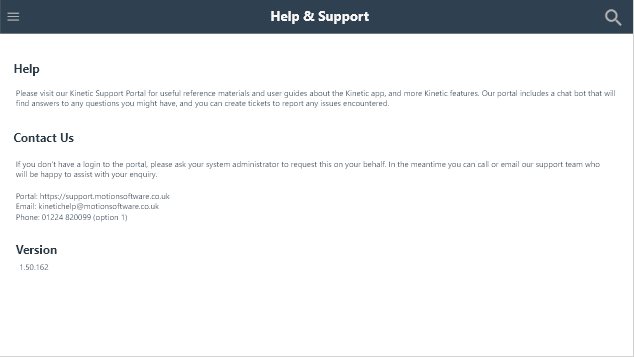
Figure 8 Query Window

If the user clicks the process button but no code has been entered, then a Xamarin popup message should appear.

­

# Help & Support

The Help & Support screen will be displayed to the user after clicking on the Help & Support option in the main menu.



On the desktop, the screen will display three headings: Help, Contact Us, and Version. Each should be in #2c3b45 bold text with the body of the text displayed under the header in regular text #2c3b45.

On the app, the screen will display three collapsible panels with headings: Help, Contact Us and Version. Each heading will be contained within a #2c3b45 bar with white text and a white chevron at the right-hand side. The bars should be clickable and can be opened and collapsed by clicking anywhere on the bar.

The sections will be: Factory Reset; Background Data Uploading; and Remote Query.

The tabs will be clickable, and a swipe gesture should be added to allow the user to swipe between the tabs. The current selected tab should have a highlighted in #00a79d bar beneath the text with unselected tabs white. The tab text will be #2c3b45.

# Appendix 1

Structure Database Data Model

All IDs will be unique to the table and auto incremental.

ApplicationInfo

{

ID,

public string Version {get; set;}

public string CustomerCode {get; set;}

public string CustomerName {get; set;}

public string ApplicationName {get; set;}

public string Service {get; set;}

}

AppSettings

{

public long ID {get; set;}

public string Name {get; set;}

public string Key {get; set;}

public string Value {get; set;}

}

ClientUploadRequest

{

public long ID {get; set;}

public string TableName {get; set;}

public string MetaData {get; set;}

public string Timestamp {get; set;}

public string Uploaded {get; set;}

public string GroupID {get; set;}

public string GroupDescription {get; set;}

public string Failed {get; set;}

public string FailReason {get; set;}

}

JobSyncStatus

{

public long ID,

public string UserId {get; set;}

public bool HasSynced {get; set;}

public string LastSynced {get; set;}

}

ErrorLog

{

public long ID {get; set;}

public string Severity {get; set;}

public string Message {get; set;}

public string Error {get; set;}

public string StackTrace {get; set;}

public string Username {get; set;}

public string UploadedToServer {get; set;}

public string LoggedDate {get; set;}

AppVersion {get; set;}

}

UploadRequestBackup

{

public long ID {get; set;}

public string Data {get; set;}

}

UserSetting

{

public long ID {get; set;}

public string SettingName {get; set;}

public string Value {get; set;}

public string Username {get; set;}

}

User

{

public long ID {get; set;}

public string KineticUsername {get; set;}

public string AccessToken {get; set;}

}

# Appendix 2

Data Database Data Model

Public class Address

{

[Key]

public string ID {get; set;}

public string Line1 {get; set;}

public string Line2 {get; set;}

public string Line3 {get; set;}

public string Line4 {get; set;}

public string City {get; set;}

public string Region {get; set;}

public string Country {get; set;}

public string Postcode {get; set;}

}

public class Contact

{

[Key]

public string ID {get; set;}

public string Name {get; set;}

public bool IsArchived {get; set;}

[ForeignKey("Address")]

public string AddressID {get; set}

public virtual Address Address {get; set;}

}

public class Customer

{

[Key]

public string ID {get; set;}

public string Name {get; set;}

public bool IsArchived {get; set;}

}

public class Data

{

[Key]

public string ID {get; set;}

public string Type {get; set;}

public string Object {get; set;}

public long SyncAuditTicks {get; set;}

public bool IsSyncRequired {get; set;}

}

public class Definition

{

[Key]

public string ID {get; set;}

public string Type {get; set;}

public string Object {get; set;}

public long SyncAuditTicks {get; set;}

}

public class Filestore

{

[Key]

public string ID {get; set;}

public string Title {get; set;}

public string Filename {get; set;}

public string FileType {get; set;}

public DateTime? LastUpdated {get;set;}

}

public class Job

{

[Key]

public string ID {get; set;}

public string No {get; set;}

public DateTime? StartDate {get; set;}

public DateTime? EndDate {get; set;}

public string Status {get; set;}

public bool IsOpen {get; set;}

public bool IsEngineerComplete {get; set;}

public DateTime? EstimatedStartDate {get; set;}

public DateTime? EstimatedEndDate {get; set;}

public decimal? EstDurationMinutes {get; set;}

public decimal? EstDurationUnits {get; set;}

public bool IsArchived {get; set;}

[ForeignKey("Site")]

public string SiteID {get; set;}

[ForeignKey("Contact")]

public string ContactID {get; set;}

[ForeignKey("Address")]

public string AddressID {get; set;}

public virtual Site Site {get; set;}

public virtual Contact Contact {get; set;}

public virtual Address Address {get; set;}

}

public class JobPerson

{

[Key]

public string ID {get; set;}

[ForeignKey("Job")]

public string JobID {get; set;}

public string PersonID {get; set;}

public virtual ICollection<Job> Job {get; set;}

}

public class LanguageText

{

[Key]

public string ID {get; set;}

public string Language {get; set;}

public string Code {get; set;}

public string Text {get; set;}

}

public class Setting

{

[Key]

public string ID {get; set;}

public string Name {get; set;}

public string Value {get; set;}

}

public class SyncAudit

{

[Key]

public string ID {get; set;}

public DateTime LastUpdated {get; set;}

public string TableName {get; set;}

public long Ticks {get; set;}

public bool IsDeleted {get; set;}

}

public class PersonSchedulerEvent

{

[Key]

public string ID {get; set}

public string PersonID {get; set;}

[ForeignKey("SchedulerEvent")]

public string SchedulerEventID {get; set;}

[ForeignKey("Job")]

public string JobID {get; set;}

public virtual ICollection<SchedulerEvent> SchedulerEvent {get; set;}

public virtual ICollection<Job> Job {get; set;}

}

public class SchedulerEvent

{

[Key]

public string ID {get; set;}

public string Title {get; set;}

public DateTime Start {get; set;}

public DateTime End {get; set;}

public string StartTime {get; set;}

public string EndTime {get; set;}

}

public class SchedulerEventItem

{

[Key]

public string ID {get; set;}

[ForeignKey("SchedulerEvent")]

public string SchedulerEventID {get; set;}

public DateTime Start {get; set;}

public DateTime End {get; set;}

public virtual SchedulerEvent SchedulerEvent {get; set;}

}

public class Site

{

[Key]

public string ID {get; set;}

public string Name {get; set}

[ForeignKey("Customer")]

public string CustomerID {get; set;}

public bool IsArchived [get; set;}

public virtual Customer Customer {get; set;}

}

public class UsergroupUser

{

public string UserID {get; set;}

public string UsergroupID {get; set;}

}

public class UsergroupPermission

{

[Key]

public string ID {get; set;}

public string UsergroupID {get; set;}

public string PermissionType {get; set;}

}

# Appendix 3 – Screen Flow

