1 FUNCTIONAL SPECIFICATION

This specification document will detail the functional requirements of the SMI Application to be custom developed to replace the Edison 360 application.

1.1 Overview of Application-Level Requirements

This document will go into fine detail regarding the requirements of this module. This section will provide a high-level overview of the requirements involved:

- User Login
  - Forgot Login (email and password)
- Navigation Menu
- Create New SMI
- Search for SMI
  - Search Results
  - Filtering and Sorting of Search Results
- User Dashboard
  - Admin Dashboard (Future for 2.0 release)
- Create A Report
- SMI Status Report

1.1.1 USER LOGIN

The login page with design notes can be viewed here in Zeplin: SMI-WF-033. The login page allows registered users to login to the app to access all the features that their account permissions gives them access to. If they type in their email and password and click submit, the credentials are validated through Active Directory and if correct they are logged in.

If the credentials are incorrect, the User will be instructed to use the Forgot Password form.

1.1.1.1 Forgot Login

The Forgot Login form can be accessed from the main Login page. There is a button with “Forgot Password?” text and upon usage, the User will see the Forgot Password form. This page can be viewed here in Zeplin: SMI-WF-034

The form will have 2 main functions. The first function will check the Active Directory database for the email that the User has entered. The output will be either an email saying the email is not found in the database and that they should contact their manager. Or the email will have instructions to fix their password settings with Microsoft Active Directory. The email content will be approved later in this process. The User will see this page upon clicking the Submit button on the previous Forgot Password page: SMI-WF-035

1.1.2 NAVIGATION MENU

The navigation menu page is the first page of the app the User will see after logging in. It shows the Bobrick logo at the top center. There are 4 menu buttons for Create New SMI, Search for SMI, User Dashboard, and Create Report. View in Zeplin: SMI-WF-036

Important to note that because this is the Navigation Menu, there is not a menu (hamburger, or 3 horizontal lines) icon to open the navigation overlay. However, you can view the Menu Icon expanded design here: SMI-WF-037

1.1.3 CREATE NEW SMI

The Create New SMI page of the app will allow the User to enter a new SMI into the system. This page will have instructions and provide a visual walkthrough via intuitive design. Required fields will be highlighted in red. Completed fields will be highlighted in green. Suggested, but optional, fields will have a standard black outline.

Preview this page in Zeplin: SMI-WF-024

Once the User fills out a red highlighted field, the field will change its border color from red to green to mark completion.

The first 3 fields will be automatically pulled from the User's profile in Active Directory. These fields are Originator, Division, and Department. Because these fields are tied to the User directly, we can have the app fill them in to reduce any app User errors and provide User confidence with the app.

The fillable fields in this page of the app are as follows:

- SMI Title, Additional Originators, Owner, Improvement Area, Description, Image Upload, Additional Documents, and Detailed Cost Savings Calculation (with Cost Savings Description)

At the end of the form, there is a large SUBMIT button. This button will be gray until all red fields have been filled in by the User. Once all fields are completed, the button will change to Bobrick blue in color.

If the User tries to use the SUBMIT button before the form is finished, the text underneath the button: "*This button will turn Bobrick blue when the form is complete.*" will be highlighted. I may also implement a popup notification, but this needs to be tested in code and approved in testing first.

When the form is complete and the SUBMIT button is Bobrick blue in color, the form will look like this: SMI-WF-024.

After the User has clicked the SUBMIT button, they will see a Thanks! page. That can be viewed in SMI-WF-040.

1.1.4 SEARCH FOR SMI

The Search for SMI page will provide the user with fillable fields to search the SMI database. None of these fields will be required and this search form will not utilize the same features as the Create SMI form (such as colored borders and a color-updating SUBMIT button). The Search for SMI page preview is here in Zeplin: SMI-WF-044

This form will have the following fields:

- SMI Number #
- Date Created and Date Modified. These will be separate and mutually exclusive when used individually. They can be used together as an advanced Search and Filter method, which is explained in the Advanced Search Methods section of the Technical Specifications. These fields will have 2 ways the User can modify to achieve a timeframe. The first will be a field with empty dates that the User can enter manually via text. The second is a calendar icon which will open a Calendar view which the User can use to select a date range. The expanded calendar date range can be viewed here in Zeplin: SMI-WF-045
    - 4 “Quick Select” buttons will autofill the date range. Those will be:
        - Annual Quarter to Date
        - Annual Year to Date
        - Fiscal Quarter to Date
        - Fiscal Year to Date
    - Radio buttons below those buttons will have “Use Date Created” and “Use Date Modified” so that the User can select which date range to view. There will also be text content letting the user know that these buttons are optional and if neither is selected, they will see both Created and Modified fields in the search results.
- The next few fields are SMI Title, Owner, Originator, Additional Originators, Improvement Area, Description, Division, and Department
- At the end of this form, there is a field titled “SMI Stage”. SMI Stage is a timeline descriptor to give the reader an idea of where in the process the SMI is. The user can choose from the following options:
    - Owner Review
    - Feasibility Assessment
    - Development
    - Requires More Information
    - Implemented
    - Not Feasible
- This can viewed on the Status Report page here: SMI-WF-048

1.1.4.1 Search Results

The Search function will pull results from the SMI database. The results will be based on the Search terms and phrases. The Search function will utilize keywords, phrases, abbreviations, and account for common misspellings.

Search will be set up to use Boolean operators such as *AND* and *OR*. Also, quotation marks for phrases.

- Usage of AND will be to limit or narrow the search to results that mention all the keywords.

- Usage of OR to broaden the search to include synonyms.
- Using quotation marks for phrases 2-4 words will be a feature.

A preview of this page can be viewed here: SMI-WF-050

1.1.4.2 Filtering and Sorting of Search Results

An extension of faceted search, a search filter is a specific attribute a visitor can use to refine the search results of a particular SMI.

- Sorting will be implemented naturally in the Excel documents.
- On the Search Results Display page, all fields from the Search form will be listed in a table format. The user will be able to click the heading to sort results by that column.
- Filtering is done based on the Search terms and conditions.

A preview of this page can be viewed here: SMI-WF-050

1.1.5 USER DASHBOARD

The User Dashboard will serve as a quick view for all the User's SMIs. The main headings will be:

- SMIs Saved to Draft (draft SMIs are deleted after 30 days)
- SMIs Submitted
- SMIs in Progress
- SMIs Implemented

Each of these headings will show the corresponding SMIs with links to their status report page. A preview of this page is here: SMI-WF-038

*An Admin Dashboard is slated for Version 2.0 with graphics, sliders, and visualized data.

1.1.6 CREATE A REPORT

A preview of this page is here: SMI-WF-049

- A report consists of information that is pulled from tables or queries, as well as information that is stored with the report design, such as labels, headings, and graphics.
- This feature will run the same as Search and the page will look the same.
- To view the Departments field expanded, visit the screen here: SMI-WF-054
- To view the Divisions field expanded, visit the screen here: SMI-WF-053

- Once the Report results are displayed, there is a field underneath that section titled “Export Report from Search Results”
- There are 2 buttons for “Export to Excel .csv” and “Export to Excel .xlsx”
    - A PDF function will be built but hidden until version 2.0 as it is not in this scope.

1.1.7 SMI STATUS REPORT

Status reports start by quickly recapping a SMIs full details. They then assess whether these SMIs are on track, at risk of falling behind schedule, or overdue. Finally, they allow Managers or Admins to edit the SMI and update the current SMI Status.

- This page also lists the notification dates and SMI Action History

User’s Status Report view can be previewed here: SMI-WF-047

The Admin view of the Status Report can be previewed here: SMI-WF-048

2 TECHNICAL SPECIFICATION

This Technical Specification describes the detailed design for implementing the SMI application.

REST, or Representational State Transfer, in the Custom Search JSON API is somewhat different from traditional REST. Instead of providing access to resources, the API provides access to a service. As a result, the API provides a single URI that acts as the service endpoint.

You can retrieve results for a particular search by sending an HTTP GET request to its URI. You pass in the details of the search request as query parameters.

2.1 Scope

This Technical Specification will address the design & implementation of the following:

- The REST API that will accept requests & provide rendered drawings in their responses.
    - Synchronous calling protocols
    - JSON request & response payloads

2.2 Exclusions

Any components and or functionality not specifically addressed in this document should not be considered as In Scope.

- pdf export

- admin dashboard

2.3 Assumptions and Dependencies

- @azure/msal-browser @azure/msal-react
- react-router-dom@5.3.3
- bootstrap react-bootstrap

"scripts"

- "build": "yarn compile && yarn next build",
- "clean": "rm -rf build-tsc .next",
- "compile": "yarn run -T tsc --build --verbose",
- "dev": "yarn compile && yarn next dev",
- "lint": "yarn run -T prettier --write './src/**/*.{js,jsx,json,ts,tsx}' && eslint './src/**/*.{js,jsx,ts,tsx}' --fix --max-warnings=0",
- "start": "yarn next start",
- "sitemap": "yarn next-sitemap"

"dependencies"

- "@headlessui/react": "^1.7.16",
- "@heroicons/react": "^2.0.18",
- "autoprefixer": "10.4.16",
- "classnames": "^2.3.2",
- "next": "^14.0.3",
- "react": "^18.2.0",
- "react-dom": "^18.2.0",
- "ts-pattern": "^5.0.4"

"devDependencies"

- "@next/eslint-plugin-next": "^14.0.3",
- "@tailwindcss/forms": "^0.5.4",
- "@tailwindcss/typography": "^0.5.9",
- "@types/node": "^20.4.5",
- "@types/react": "^18.2.18",
- "@types/tailwindcss": "^3.1.0",
- "@types/webpack-env": "^1.18.1",
- "@typescript-eslint/eslint-plugin": "^6.2.1",
- "@typescript-eslint/parser": "^6.2.1",
- "cssnano": "^6.0.1",

```
-       "eslint": "^8.46.0",
-       "eslint-config-next": "^14.0.3",
-       "eslint-config-prettier": "^9.1.0",
-       "eslint-plugin-import": "^2.28.0",
-       "eslint-plugin-jsx-a11y": "^6.7.1",
-       "eslint-plugin-module-resolver": "^1.5.0",
-       "eslint-plugin-monorepo": "^0.3.2",
-       "eslint-plugin-react": "^7.33.1",
-       "eslint-plugin-react-hooks": "^4.6.0",
-       "eslint-plugin-react-memo": "^0.0.3",
-       "eslint-plugin-react-native": "^4.0.0",
-       "eslint-plugin-simple-import-sort": "^10.0.0",
-       "eslint-plugin-sort-keys-fix": "^1.1.2",
-       "postcss": "^8.4.27",
-       "postcss-preset-env": "^9.1.0",
-       "prettier": "^3.0.0",
-       "prettier-plugin-tailwindcss": "^0.5.7",
-       "sass": "^1.64.2",
-       "sort-package-json": "^2.5.1",
-       "stylelint": "^15.10.2",
-       "stylelint-config-recommended": "^13.0.0",
-       "stylelint-order": "^6.0.3",
-       "stylelint-prettier": "^4.0.2",
-       "tailwindcss": "^3.3.3",
-       "typescript": "5.3.2"
-     "resolutions"
-       "autoprefixer": "10.4.5"
```

2.4 REST API

2.4.1 REST API URLS

The REST API URLs to be supported are as follows:

GET /_api/search/query	GET http://server/_api/search/query?query_parameter=value&query_parameter=value GET http://server/_api/search/query(query_parameter=value&query_parameter=<value>)
POST /_api/search/postquery	For POST requests, you pass the query parameters in the request in JavaScript Object Notation (JSON) format. The HTTP POST version of the Search REST service supports all parameters supported by the HTTP GET version.
PUT	The PUT method is used to update a resource on the server. The resource is identified by a URI and the updated data is sent in the body of the request.
DELETE (This will be an Admin-only feature)	The DELETE method is used to delete a resource on the server. The resource is identified by a URI and the server removes the resource permanently.

2.4.2 REST API JSON PAYLOAD

JavaScript has built-in methods to encode and decode JSON either through the Fetch API or another HTTP client. Server-side technologies have libraries that can decode JSON without doing much work.

To make sure that when our REST API app responds with JSON that clients interpret it as such, we should set Content-Type in the response header to application/json after the request is made. Many server-side app frameworks set the response header automatically. Some HTTP clients look at the Content-Type response header and parse the data according to that format.

This example will use the Express back end framework for Node.js. We can use the body-parser middleware to parse the JSON request body, and then we can call the res.json method with the object that we want to return as the JSON response as follows:

```
const express = require('express');
const bodyParser = require('body-parser');

const app = express();
```

Commented [1]: @Jeremy Parker - This is one of the most important parts of this document to me, so that we can define exactly what our back-end team will need to implement to support the React app. There will need to be a number of well-defined REST APIs, which you said React could provide, so I'll want to include those. I'm ready to start lining up a resource to take care of this for us, so I want them to be able to understand what we need from them so they can cost out their work as accurately as possible. We'll review later today.

```
app.use(bodyParser.json());

app.post('/', (req, res) => {
  res.json(req.body);
});

app.listen(3000, () => console.log('server started'));
```

bodyParser.json() parses the JSON request body string into a JavaScript object and then assigns it to the req.body object.

Set the Content-Type header in the response to application/json; charset=utf-8 without any changes. The method above applies to most other back end frameworks.

GET retrieves resources.

POST submits new data to the server.

PUT updates existing data.

DELETE removes data.

```
const express = require('express');
const bodyParser = require('body-parser');

const app = express();

app.use(bodyParser.json());

app.get('/smi', (req, res) => {
  const smi  = [];
  // code to retrieve an smi...
  res.json(smi);
});

app.post('/smi', (req, res) => {
  // code to add a new smi...
  res.json(req.body);
});

app.put('/smi/:id', (req, res) => {
  const { id } = req.params;
  // code to update an smi...
```

```
  res.json(req.body);
});

app.delete('/smi/:id', (req, res) => {
  const { id } = req.params;
  // code to delete an smi...
  res.json({ deleted: id });
});
app.listen(3000, () => console.log('server started'));
```

The POST, PUT, and DELETE endpoints all take JSON as the request body, and they all return JSON as the response, including the GET endpoint.

2.4.3 ERROR HANDLING

To eliminate confusion for API users when an error occurs, we should handle errors gracefully and return HTTP response codes that indicate what kind of error occurred. This gives maintainers of the API enough information to understand the problem that's occurred. We don't want errors to bring down our system, so we can leave them unhandled, which means that the API consumer has to handle them.

Common error HTTP status codes include:

- 400 Bad Request - This means that client-side input fails validation.
- 401 Unauthorized - This means the user isn't authorized to access a resource. It usually returns when the user isn't authenticated. We will use this as a permissions error page.
- 403 Forbidden - This means the user is authenticated, but it's not allowed to access a resource. We will use this as a permissions error page.
- 404 Not Found - This indicates that a resource is not found.
- 500 Internal server error - This is a generic server error. It probably shouldn't be thrown explicitly.
- 502 Bad Gateway - This indicates an invalid response from an upstream server.
- 503 Service Unavailable - This indicates that something unexpected happened on server side (It can be anything like server overload, some parts of the system failed, etc.).

We also need ways to paginate data so that we only return a few results at a time. We don't want to tie up resources for too long by trying to get all the requested data at once.

Filtering and pagination both increase performance by reducing the usage of server resources. As more data accumulates in the database, the more important these features become.

2.4.4 CONSUME REST API'S IN REACT

useEffect Hook: In React, we perform API requests within the useEffect() hook. It either renders immediately when the app mounts or after a specific state is reached. This is the general syntax we'll use:

```
useEffect(() => {
    // data fetching here
}, []);
```

useState Hook: When we request data, we must prepare a state in which the data will be stored when it is returned. We can save it in a state management tool such as Redux or in a context object. To keep things simple, we'll store the returned data in the React local state.

```
const [posts, setPosts] = useState([]);
```

2.4.5 CONSUME APIS USING THE FETCH API

The Fetch API is a JavaScript built-in method for retrieving resources from a server or an API endpoint. It's built-in, so you don't need to install any dependencies or packages.

The fetch() method requires a mandatory argument, which is the path or URL to the resource you want to fetch. Then it returns a Promise so you can handle success or failure using the then() and catch() methods.

We are fetching data from a URL that returns data as JSON and then logging it to the console:

```
fetch('https://placeholder.internet/posts?_limit=10')
  .then(response => response.json())
  .then(data => console.log(data));
```

The default response is usually a regular HTTP response rather than the actual JSON, but we can get our output as a JSON object by using the response's json() method.

GET Request in React with Fetch API

We can use the HTTP GET method to request data from an endpoint.

For other methods such as POST and DELETE, we'll attach the method to the options array:

```
fetch(url, {
    method: "GET" // default, so we can ignore
})
```

2.4.6 AXIOS JAVASCRIPT LIBRARY

This could be a backup to the FETCH API. AXIOS is great for HTTP requests, but Fetch API is more modern version of the AXIOS api. I'm listing it here as I will likely use it for exporting to excel and reporting.

Axios makes it easy to send asynchronous HTTP requests to REST endpoints and perform CRUD operations (create, read, update and delete), as well as handle the responses.

Below is an example using placeholder and posts. Posts will be the SMIs in this instance.

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

function App() {
  const [posts, setPosts] = useState([]);

  useEffect(() => {
    axios.get('https://jsonplaceholder.typicode.com/posts')
      .then(response => {
        setPosts(response.data);
      })
      .catch(error => {
        console.error(error);
      });
  }, []);

  return (
    <ul>
      {posts.map(post => (
        <li key={post.id}>{post.title}</li>
      ))}
    </ul>
  );
}
```

2.4.7 GRAPHQL AND REACT

One of the main benefits of GraphQL is the User's ability to request what they need from the server and receive that data exactly and predictably.

A GraphQL client for React using modern context and hooks APIs that's lightweight (< 4 kB) but powerful; the first Relay and Apollo alternative with server side rendering.

The exports can also be used to custom load, cache and server side render any data, even from non-GraphQL sources.

2.4.8 EXPORT TO EXCEL FILES (.CSV AND .XLSX)

```
import { useEffect, useState } from "react";
import "./App.css";
import * as XLSX from "xlsx";

function App() {
  const [products, setProducts] = useState([]);
useEffect(() => {
    fetch("SMI link")
      .then((res) => res.json())
      .then((json) => setProducts(json));
 }, []);

  const handleDownload = () => {
    // flatten object like this {id: 1, title:'', category: ''};
    const rows = products.map((product) => ({
      id: smmi.id,
      title: smi.title,
      category: smi.category,
    }));

    // create workbook and worksheet
    const workbook = XLSX.utils.book_new();
    const worksheet = XLSX.utils.json_to_sheet(rows);

    XLSX.utils.book_append_sheet(workbook, worksheet, "SMIs");
```

```
    // customize header names
    XLSX.utils.sheet_add_aoa(worksheet, [
      ["Product ID", "Product Name", "Product Category"],
    ]);

    XLSX.writeFile(workbook, "ReportFor2023.xlsx", { compression: true });
  };
  return (
    <div className="wrapper">
      <button onClick={handleDownload}>DOWNLOAD EXCEL</button>
    </div>
  );
}
```

2.4.9 STALE-WHILE-REVALIDATE (SWR)

This method is used to fetch data from a server and is used in React. It manages any issues that may arise when obtaining the data and helps you manage its storage. SWR includes useState() and useEffect(), so there is no need to import them.

The advantages of SWR

SWR speeds up the app's loading time by showing older data while fetching the latest information.

It reduces the server burden by minimizing the number of requests.

Even if there is a bad connection, or no connection at all, SWR can still display previously fetched data.

SWR handles data acquisition and maintenance without the use of sophisticated coding.

It knows what to do if something goes wrong while gathering data.

We can change how SWR operates to better suit your app.

It provides a consistent approach to collecting and saving data across the app.

3 SCREEN DESIGN

Screen designs & screen-level requirements will go here. Screens should be incorporated by Figma/Zeplin reference number; it is not necessary or desired to copy & paste screen images here, since they will be finalized & approved in Zeplin. (add prototype link)

3.1 SMI Login Page (SMI-WF-033)

- This page will be known as the Access page on the html site. It is part of the Login family. The User will always see this page first when accessing the site and will need to successfully login to access the application.
- Bobrick branding on landing screen.
- The use of “Bobrick blue” throughout the app in both light and heavy applications
- Feature function to save login information to device.
- Icon usage in each field for a familiar feel.
- Large Submit button helps display confidence and strength to the user, or it’s a one size fits all to any users’ tapping tool.
- Copyright will display on the Login family pages only

3.2 Forgot Password Page (SMI-WF-034) & Reset Instructions (SMI-WF-035)

- This page will be known as the Forgot page on the html site. It is part of the Login family.
- The User will see this after tapping the Forgot Password button.
- Description of next step after user taps Submit. “*Check your email for a Password Reset Link*”
- Placed directly under the SUBMIT button to utilize Users’ eye focus.
- Password Reset Email sent to user with link to reset
- Feature function button to quickly return to the login screen.
- The user can also use the swipe feature on recent devices to go back.

3.3 Main Menu after login (SMI-WF-036) & icon Nav Menu (SMI-WF-037)

- This page will be known as the Main Menu page on the html site. It is part of the Login family.

- The User will see this after successfully completing login.
- These are the four main functions of the app.
- This button will only show to those User's with administrative permissions.

3.4 User Dashboard (SMI-WF-038)

- This User Dashboard is the Mobile version.
- There is no Manager or Admin function in Mobile view. Those actions are saved for the Tablet/PC view utilizing a full screen, 2 column display.
- For Draft SMIs, Users will have 30 days to Submit or the SMI will be auto-deleted.
    - Green will signify an SMI under 10 days old
    - Orange will mean 10+ days (10-19 days old)
    - Red will mean 20+ days (20-30 days old)
    - Notifications will go out for Orange, Red, and Deleted SMIs. Deleted SMIs cannot be recovered.
- The Submitted section will show SMIs that the User has submitted.
- For a full list of every SMI ever created by this User, they will need to run a report, OR they can view everything on a tablet/PC screen size view.
    - An expanded dashboard with the full User list and charting tools are planned for a 2.0 update
- SMIs that are in progress are either in progress or in a manager/admin queue for special review
- SMIs that are closed will show a black link but still be active when tapped

3.5 Create New SMI (SMI-WF-039)

- This dash outlined box will disappear after 3 logins.
    - Required fields are outlined in RED
    - Completed fields are outlined in GREEN
    - Suggested fields are outlined in BLACK
- These are visual cue descriptions to guide the user through the app usage.
- They are app wide for any text fields.
- The app will pull the User's data and display the top-level fields.
- The Additional Originator(s), field is not a required field because the logged in User counts as the main Originator
    - The " (s), " is to intuitively present the options for multiple names separated by a comma.
    - There is a smaller bit of instruction text included as well for the comma usage.

3.6 Thanks! Your SMI has been created. (SMI-WF-040)

- Thank you confirmation page after user Submits their new SMI
- 2 ways to navigate away from this screen, icon nav and the Go To User Dashboard button

3.7 Improvement Area expanded (SMI-WF-041)

- Displaying what the User will see after tapping the Select button in the Improvement Area field.
- These are the updated categories for this field.
- The Tap twice to Select function gives the User 2 options:
    - one to miss-tap once and,
    - another to utilize the drag-&-drop or tap-&-drop feature.

3.8 Browse Device for file upload preview (SMI-WF-042)

- Example window showing what the User will see when they access their device to upload a file. Views will differ per device
- Users will see these fields populate with uploaded files as they are added to the SMI.
- Both the Image field and the Additional Documents field are not required so the field outlines will always be neutral black.
- The SUBMIT button turns Bobrick blue when all fields are completed and this form is ready to submit.

3.9 Error notification box / pop-up for unfinished form (SMI-WF-043)

- This notification box or “Alert” box pops up over a grayed-out display when the User clicks the Submit button before all fields have been completed.
- The notification will float with the screen so it will always appear in view and not at the top as this example shows.

3.10 Search for SMI (SMI-WF-044)

- 2 date fields to give User access to more timeline points
- These fields are mutually exclusive search parameters (testing will be done to make these mutually inclusive, but currently nothing exists to search for 2 date ranges, so will have to build it)
- Buttons to quick select blocks of time, Fiscal year is November 1 thru Oct 31

- Q1 - Nov - Jan
  Q2 - Feb - April
  Q3 - May - July
  Q4 - Aug – Oct

- Only 1 field is required to be filled out in order for Search to function.
- Search will be configured to use 'AND' criteria commands. (eg. keyword AND keyword AND keyword)
- Results will prioritize active SMIs
- User can click headers of each column as a sorting feature
- Large Bobrick blue Search button
- Clear Search Fields button

3.11 Search for SMI – Date Range Expanded (SMI-WF-045)

- Example of the expanded calendar view when User taps either the calendar icon or the date input field

3.12 SMI Search Results (SMI-WF-046)

- Example of the Search Results display
- Result expanded in a quick view.
- Open Full SMI Details Button at bottom of overlay window to bring User to the SMI Status page.
- Revise Search button to return to a filled-out form.
- Text option to clear search fields will be available above all fields

3.13 SMI Status Report (SMI-WF-047)

- Full SMI details with SMI Status Report title.
- This area will show the updates made from SMI creation to Implemented to Closed.
- No changes can be made at the User permission level.
- Full action history

3.14 SMI Status Report – ADMIN (SMI-WF-048)

- This is what those with Administrative permissions will see.
- The mobile version does not allow any user to edit SMIs, this action must be done in the PC view.
- The mobile version will allow a Manager or Admin to Change SMI Status.
- Option to Change SMI Status from a drop-down menu
- Notes for Admin to explain the change of status.

3.15 Create Report (SMI-WF-049)

- This screen is intentionally designed to look, feel, and function like Search and Search Results
- These buttons will autofill the dates based off of Bobrick's Fiscal Quarter dates
- Search will function with at least 1 completed field.
- Additional fields will help with the AND search function
    - For example: Owner AND SMI Stage AND Fiscal YTD for narrower results.

3.16 Report Results and Export (SMI-WF-050)

- This page will only show to Users on window size Tablet and PC.
- This allows the app to display properly with all feature functions including column sorting and exporting to an Excel file type.
- Users who use the Report page on Mobile will see Search Results but will not have the table data or the export buttons. There will be a notification for this and text instructions on Search Results page.
- Search Query to serve as visual and memory cue.
- Export buttons with both clear options.

3.17 Prototype-only Example Pages (SMI-WF-051 – SMI-WF-054)

SMI-WF-051 and SMI-WF-052: Prototype-lead pages

SMI-WF-053: Page showing Divisions expanded

- The pop-out field is outlined in RED to indicate that the field is not complete. This will adopt the other form outline colors for instruction and completion.
- The divisions list is currently 9 entries. Easily separated into 3 columns of 3
- The DONE button mimics the grayed-out SUBMIT button and will turn Bobrick blue once a selection has been made.

- Departments list is currently 69 entries
    - This is sorted into 5 columns that are mostly full. Anything more or less would create negative space and/or more columns
- The pop-out field is outlined in RED to indicate that the field is not complete. This will adopt the other form outline colors for instruction and completion.
- The DONE button mimics the grayed-out SUBMIT button and will turn Bobrick blue once a selection has been made.
- Instead of a “X” or a checkmark ”✓”, I’ve designed this to fill the checkbox with a Bobrick blue.
    - This creates higher visibility rather than a black & white text-filled screen with older looking elements such as the X or ✓
    - Once at least 1 box is marked Bobrick blue, then the outline will follow formula and turn green.
    - The DONE button will also change to Bobrick blue.