

REPORT CREATION – TECHNICAL LLD

This document has been approved by

|  |  |
| --- | --- |
|  |  |
| IFS Business consultant | Customer Solution owner or accountable role |
|  |  |
| Date | Date |

Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Author | Remarks |
| A1 | 2023-12-21 | DILIP | I have created the Technical LLD document creation and will be submitted for review to the team. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1 Report Creation Purpose and Guideance 4](#_Toc152000658)

[2 Functional Solution 6](#_Toc152000659)

[3 Business Requirements and References 4](#_Toc152000660)

[4 Development Solution 4](#_Toc152000668)

[4.3.1 Functional Requirement and Prerequisites 4](#_Toc152000669)

[4.3.2 Functional Solution 4](#_Toc152000670)

[4.3.3 Development Specification 4](#_Toc152000671)

[5 User Flow Digrams 4](#_Toc152000672)

[6 Wireframes 4](#_Toc152000673)

DEVELOPMENT PURPOSE AND GUIDANCE

This component introduces an interface to create preliminary reports for Expert Advisory Menu Card Services. This tool will take a Jira Issue ID as input and automatically gather essential information from various sources such as Jira, Service Now, SharePoint, Power BI and the IFS Digital Assist database to generate a draft of the final report.

The draft is based on the final report template. When a new template is created Project Team needs to be notified of the new report template. The project team will take in the new final report template and do the necessary adjustments to make it compatible with Report Creation tool. The report creation tool will pick the latest available final report template from its own repository that is managed by the project team.

Functional Requirement and Prerequisites

The functional requirements outlined above for user authentication, data integration, and report creation set the foundation for developing a robust and user-friendly Report Creation Tool. These requirements ensure that the tool securely authenticates users, integrates data from multiple sources, and provides an effective interface for creating preliminary reports for Expert Advisory Menu Card Services.

## **User Authentication**

### Purpose

The user authentication functionality is essential to ensure secure access to the Report Creation Tool. Azure AD Authentication will be implemented to allow users to log in using their Azure AD accounts, ensuring personalized access to assigned Jira issues and facilitating a streamlined report creation process.

### Functional Requirements

1. **Azure AD Authentication:**
   * Users must log in using their Azure AD accounts.
   * The authentication process should fetch basic user information such as Name, Email, and Designation from Azure AD.
2. **Personalized Access:**
   * Users should only have access to Jira issues assigned to them.
   * The tool should differentiate between open and closed Jira issues for the logged-in user.
3. **Secure Login System:**
   * MSAL (Microsoft Authentication Library) will be utilized for secure login both on the frontend (ReactJS) and backend (Django).

## **Data Integration**

### Purpose

Data integration is crucial for gathering information from various sources to generate a comprehensive draft report. The integration will include fetching details from Jira, Service Now, SharePoint, and the IFS Digital Assyst Database.

### Functional Requirements

1. **Jira Integration:**
   * The tool should fetch Jira issue details, including Jira ID, Sub Task details, Customer Name, Menu Card ID, Menu Card Description, Ticket Description, SDO, SDM, CSM, Creator, Assigned Date, Created On, and Report Status.
2. **Service Now, SharePoint, Power BI, and IFS Database Integration:**
   * Integration with Service Now, SharePoint, Power BI, and IFS Digital Assyst Database to collect additional information required for report generation, such as Action Items and related details like Item ID.

## **Report Creation**

### Purpose

The Report Creation functionality allows users to interact with the tool, view assigned Jira issues, and generate draft reports based on a template. It involves creating a user-friendly interface with various screens to facilitate the report creation process.

### Functional Requirements

1. **List of Assigned Jira Issues:**
   * Users should be able to view a list of Jira issues assigned to them, differentiating between open and closed issues.
2. **Detail Page of Jira Issue:**
   * The tool should provide a detailed view of a specific Jira issue, including Action Items and associated details.
3. **Action Items:**
   * For each Jira issue, the tool should display Action Items with relevant information, including Item ID, Jira ID, Sub Task, Customer Name, Menu Card ID, Menu Card Description, Ticket Description, SDO, SDM, CSM, Creator, Assigned Date, Created On, and Report Status.
4. **Additional Features:**
   * Subtask Item Description: Include subtask item description for clarity.
   * Updated/Refreshed Status: Consider adding a status to identify if the record is "Updated" or "Refreshed."
   * History Logs: Implement a feature to track changes and updates, providing a comprehensive history of actions taken on each record.

Functional Solution

## **Overview**

The functional solution outlined ensures the successful development of the Report Creation Tool. Leveraging technologies such as Django, React with Vite, MySQL, and MSAL, the solution addresses key business requirements, providing a secure, integrated, and user-friendly platform for generating preliminary reports for Expert Advisory Menu Card Services. The development process, including version control, CI/CD, testing, and documentation, ensures the delivery of a robust and reliable tool.

## **Technology Stack**

1. ****Backend API: Django****
   * Django will serve as the backend framework, providing APIs for fetching Jira issues and relevant data from the integrated sources.

Here's a brief overview of the purpose of some of the listed packages:

* **amqp==5.2.0**: AMQP (Advanced Message Queuing Protocol) is a messaging protocol. This package likely provides support for AMQP in your application.
* **asgiref==3.7.2**: ASGI (Asynchronous Server Gateway Interface) is a specification for asynchronous web servers. This package provides a reference implementation for ASGI.
* **Babel==2.13.1**: Babel is used for internationalization and localization in your application.
* **billiard==4.2.0**: This is a Python multiprocessing fork with improvements and bug fixes.
* **celery==5.3.6**: Celery is a distributed task queue system. It's commonly used for background task processing in web applications.
* **certifi==2023.11.17**: Certifi provides a carefully curated collection of Root Certificates for validating the trustworthiness of SSL certificates.
* **Django==4.2.7**: Django is a high-level Python web framework. It simplifies the process of building web applications by providing reusable components.
* **djangorestframework==3.14.0**: Django Rest Framework is a powerful and flexible toolkit for building Web APIs on top of Django.
* **drf-yasg==1.21.7**: This package generates real Swagger/OpenAPI 2.0 specifications from a Django Rest Framework API.
* **Jinja2==3.1.2**: Jinja2 is a template engine for Python. It's often used in web frameworks like Flask and Django.
* **kombu==5.3.4**: Kombu is a messaging library for Python that supports multiple message brokers, including RabbitMQ.
* **lxml==4.9.3**: Lxml is a library for processing XML and HTML. It's often used for web scraping and parsing XML documents.
* **mysqlclient==2.2.0**: This package provides a MySQL database connector for Python.
* **redis==5.0.1**: Redis is an in-memory data structure store. This package likely provides Python bindings for interacting with a Redis server.
* **requests==2.31.0**: Requests is a popular HTTP library for making HTTP requests in Python.
* **sqlparse==0.4.4**: Sqlparse is a non-validating SQL parser. It can be useful for formatting SQL queries.

1. ****Frontend: React with Vite****
   * The frontend will be developed using React with Vite as the build tool, offering a responsive and performing user interface.

Here's a brief overview of list with package versions included:

* **@azure/msal-browser (^3.6.0)**: Microsoft Authentication Library for JavaScript (MSAL) - Browser version.
* **@azure/msal-react (^2.0.8)**: Microsoft Authentication Library for React - React bindings for MSAL.
* **@emotion/react (^11.11.1)**: React integration for Emotion, a popular CSS-in-JS library.
* **@emotion/styled (^11.11.0)**: Styled components for Emotion, providing a way to style React components.
* **@mui/icons-material (^5.15.0)**: Material-UI Icons - Icons component for Material-UI.
* **@mui/material (^5.15.0)**: Material-UI - A popular React UI framework based on Google's Material Design.
* **@reduxjs/toolkit (^1.9.7)**: Redux Toolkit - A set of tools to simplify Redux development.
* **apexcharts (^3.44.2)**: Interactive JavaScript charts library using SVG, providing a wide range of customizable charts.
* **axios (^1.6.2)**: Promise-based HTTP client for the browser and Node.js, enabling easy communication with APIs.
* **bootstrap (^5.3.2)**: A popular CSS framework for building responsive and mobile-first websites.
* **react (^18.2.0)**: A JavaScript library for building user interfaces.
* **react-apexcharts (^1.4.1)**: React wrapper for ApexCharts, making it easy to integrate interactive charts into React applications.
* **react-dom (^18.2.0)**: Entry point to the DOM and server renderers for React.
* **react-helmet-async (^2.0.3)**: Asynchronous version of React Helmet for managing document head in React.
* **react-router-dom (^6.20.1)**: Declarative routing for React applications, enabling navigation between different views.
* **sass (^1.69.5)**: CSS preprocessor scripting language, extending CSS with features like variables and nesting.

1. **Database: MySQL**
   * MySQL will be used as the database to store master data, including information about Service Delivery Officers (SDOs), Customer Service Managers (CSMs), Service Delivery Managers (SDMs), projects, customers, and menu cards.

Database Structure Link :

1. **Authentication: Microsoft Authentication Library (MSAL)**
   * MSAL will be implemented for secure user authentication, ensuring a seamless and secure login process.

## **Functional Components**

### User Authentication

1. **Azure AD Authentication:**
   * Users will log in using their Azure AD accounts.
   * Azure AD Authentication will fetch basic user information such as Name, Email, and Designation.
2. **Personalized Access:**
   * The system will grant users access only to Jira issues assigned to them.
   * The tool will differentiate between open and closed Jira issues for the logged-in user.

### 3.3.2 Data Integration

1. **Jira Integration:**
   * The tool will fetch comprehensive details for each Jira issue, including Jira ID, Sub Task information, Customer Name, Menu Card ID, Menu Card Description, Ticket Description, SDO, SDM, CSM, Creator, Assigned Date, Created On, and Report Status.
2. **Other Source Integration:**
   * Integration with Service Now, SharePoint, Power BI, and IFS Digital Assyst Database will provide additional essential information for report generation, such as Action Items and related details.

### Report Creation

1. **List of Assigned Jira Issues:**
   * Users will have access to a list of Jira issues assigned to them, distinguishing between open and closed issues.
2. **Detail Page of Jira Issue:**
   * The tool will provide a detailed view of a specific Jira issue, including Action Items and associated details.
3. **Action Items:**
   * For each Jira issue, the tool will display Action Items with relevant information, including Item ID, Jira ID, Sub Task, Customer Name, Menu Card ID, Menu Card Description, Ticket Description, SDO, SDM, CSM, Creator, Assigned Date, Created On, and Report Status.
4. **Additional Features:**
   * Subtask Item Description: Include subtask item description for clarity.
   * Updated/Refreshed Status: Consider adding a status to identify if the record is "Updated" or "Refreshed."
   * History Logs: Implement a feature to track changes and updates, providing a comprehensive history of actions taken on each record.

## **Development Process**

1. **Version Control: Git**
   * Git will be used for version control, enabling efficient collaboration and code management.
2. **Continuous Integration and Deployment (CI/CD):**
   * CI/CD pipelines will be established for automated testing, building, and deployment, ensuring a streamlined development process.

**Development Specification**

### 4.1 Login Page

#### 4.1.1 Purpose

The Login Page in ReactJS, part of the Success Pilot project, is designed for secure user authentication using Azure AD accounts.

### **ReactJS (Frontend - Success Pilot)**

### Development Specifications

1. **Components:**
   * Create a Login component within the Account app.
   * Utilize MSAL for user authentication and retrieval of basic user information.
2. **User Interface:**
   * Develop an intuitive login form within the Account app.
   * Implement error handling for authentication failures.
3. **Integration:**
   * Connect the Login component with the backend Django API endpoint for Azure AD authentication

### BASIC Folder structure

React JS Advanced Folder Structure

│ .env

│ .eslintrc.cjs

│ .gitignore

│ index.html

│ package-lock.json

│ package.json

│ README.md

│ vite.config.js

│

├───public

│ vite.svg

│

└───src

│ App.css

│ App.jsx

│ index.css

│ main.jsx

│

├───assets

│ ├───Images

│ │ ad\_icon.svg

│ │ images.js

│ │ login\_bg.jpg

│ │ login\_icon.svg

│ │ Logo.svg

│ │ patner-icon.png

│ │ profile.jpg

│ │ star.PNG

│ │ sub-task.png

│ │

│ └───scss

│ │ styles.scss

│ │

│ └───components

│ \_issueDetails.scss

│ \_lisiting.scss

│ \_loader.scss

│ \_login.scss

│

├───axiosInstance

│ axiosInstance .js

│

├───components

│ │ IconBreadcrumbs.jsx

│ │

│ ├───account

│ │ UserProfile.jsx

│ │

│ ├───IssueListing

│ │ │ AssignedIssueListing.jsx

│ │ │ HistoricReport.jsx

│ │ │ IssueBody.jsx

│ │ │ IssueDetailsPage.jsx

│ │ │ IssueList.jsx

│ │ │ OngoingReport.jsx

│ │ │

│ │ └───createReport

│ │ CreateReportContent.jsx

│ │ ImageUpload.jsx

│ │

│ ├───navabar

│ │ SideBar.jsx

│ │ TopNav.jsx

│ │

│ ├───shared

│ │ └───common

│ │ CustomModal.jsx

│ │ CustomTabs.jsx

│ │ DrawerComponent.jsx

│ │ Dropdown.jsx

│ │ ImageElement.jsx

│ │ Loader.jsx

│ │ Seo.jsx

│ │ TableHead.jsx

│ │

│ └───statistics

│ ChartData.js

│ Statistics.jsx

│

├───config

│ authConfig.js

│

├───constants

│ number.js

│ routes.js

│ static.js

│

├───pages

│ ├───auth

│ │ Auth copy.jsx

│ │ Auth.jsx

│ │

│ └───home

│ Home.jsx

│

├───routes

│ AppRoutes.jsx

│

└───utils

NavigationClient.js

Here's what each file and directory does:

* node\_modules/: This directory contains all the npm packages required for your project.
* public/: This directory contains static assets that are not processed by Vite.
* public/index.html: This is the HTML file that will be served by Vite.
* src/: This directory contains the source code for your React application.
* src/main.jsx: This is the entry point for your React application.
* src/App.jsx: This is the main React component for your application.
* .gitignore: This file specifies the files and directories that Git should ignore.
* package.json: This file contains information about your project and its dependencies.
* vite.config.js: This is the configuration file for Vite.

**Assets :** All images and css managed in assets folder.Anone can change css of an component and images if require to update or change.

assets/images :

logo.svg – logo image

login\_icon.svg – Login icon

partner-icon.png – partner icon used in issue listing page

sub-task.png – sub task icon used in issue listing page

assets/scss :

style.scss – The `@import` rule in SCSS is used to import other SCSS files into the current file. In this case, it is importing two files: 'components/listing' and 'components/login'. This means that the styles defined in those files will be available in the current file, allowing you to use any variables, mixins, or styles defined in those files.

\_listing.scss – This file having all css related to listing page.

For different status used below css :

// create report status css

/\* Shared styles for all status \*/

.report-status {

  padding: 5px 10px;

  border-radius: 5px;

  font-weight: bold;

}

/\* Styles for 'Not Created' \*/

.report-status.status-0 {

  background-color: #FFA500; /\* Orange \*/

  color: #FFF; /\* White \*/

}

/\* Styles for 'Creating Report' \*/

.report-status.status-1 {

  background-color: #87CEEB; /\* Sky Blue \*/

  color: #000; /\* Black \*/

}

/\* Styles for 'Created' \*/

.report-status.status-2 {

  background-color: #4bd62f; /\* Red \*/

  color: #FFF; /\* White \*/

}

/\* Styles for 'Creation Error' \*/

.report-status.status-3 {

  background-color: #ff523b; /\* Green \*/

  color: #FFF; /\* White \*/

}

\_issueDetails.scss – This css related to issue detail page.

// css for ongoing report

.report-title {

  font-weight: bold;

}

.report-In-progress {

  font-weight: bold;

  padding: 5px;

  background-color: rgb(101, 241, 101);

  border-radius: 5px;

}

.ticket-des {

  padding: 0px;

  margin: 0px;

}

\_login.scss – This css related to login page.

body {

  font-family: 'Roboto', sans-serif !important;

}

.login\_container {

  width: 600px;

  text-align: center;

  padding: 80px;

  background: #fff;

  display: flex;

  align-items: center;

}

.login-azure-btn {

  margin-top: 20px;

  width: 100%;

  background-color: #503998;

  color: white;

  border-radius: 0px;

  font-weight: 700;

  margin-bottom: 20px;

  padding: 10px;

  border: none;

}

.login-azure-btn:hover {

  background-color: #544879;

  border-radius: 0px;

  font-weight: 700;

}

.MuiBreadcrumbs-separator {

  margin: 2px !important;

}

\_loader.scss – css used for loader.

**Main Components:**

IconBreadcrumbs – This component is used for Breadcrumbs management.

Account/ UserProfile – To Manage user profile

axiosInstance/ axiosInstance – This component used for API call.

IssueListing/ AssignedIssueListing – This components is used skelton for jira issue listing assigned to login user.

IssueListing/ HistoricReport – This will show history related to issue.

IssueListing/ IssueBody – This component having main body of issue listing.

IssueListing/ IssueDetailsPage – Detail information of issue like client logo, name etc.

IssueListing/ IssueList – This is the main component of jira issue list page where API is called and data pass to IssueBody component.

IssueListing/ CreateReport/ CreateReportContent – CreateReport component is used for report creation and mange all element of pop up of Create Report.

IssueListing/ CreateReport/ CreateReportContent – supporting to upload image for create report.

navabar / SideBar – SideBar component is used to manage left side link/menu.

navabar / TopNav – TopNav is used to manage top menu or links.

Shared/common : All common shareable components used by different component.

shared/common/CustomModel – This is the common model used for model/pop like create report

shared/common/Seo– Used for title of application.

shared/common/CustomTabs –Used to manage Tabs in application.

shared/common/Drawer Component– Used to manage Drawer

shared/common/IconBreadcrumbs–Managing Breadcrumbs

shared/common/Dropdown–Used for dropdown/list box elements

shared/common/Loader–Used to mange loader element in app.

shared/common/TableHead– Used to manage table heads.

config/authConfig – Used to keep credential of msal login.

constants/number – This component containing all number to follow sonar standards.

constants/routes – This file used to keep routes name given.

export const routePath = {

  DASHBOARD: '/dashboard',

  LOGIN: '/',

  HOME: '/',

  ISSUE\_LISTING: '/issue-listing',

  ISSUEBYID: '/issue-details/'

}

constants/static – Manage static names.

export const issueListTableHeaders = [

  { label: '#', className: '', scope: 'col' },

  { label: import.meta.env.VITE\_JIRA\_ID, className: '', scope: 'col' },

  { label: import.meta.env.VITE\_CUSTOMER\_NAME, className: '', scope: 'col' },

  { label: import.meta.env.VITE\_MENU\_CARD\_ID, className: '', scope: 'col' },

  { label: import.meta.env.VITE\_MENU\_CARD\_DESCRIPTION, className: '', scope: 'col' },

  { label: import.meta.env.VITE\_TICKET\_DESCRIPTION, className: '', scope: 'col' },

  { label: import.meta.env.VITE\_SDO, className: '', scope: 'col' },

  { label: import.meta.env.VITE\_SDM, className: '', scope: 'col' },

  { label: import.meta.env.VITE\_CSM, className: '', scope: 'col' },

  { label: import.meta.env.VITE\_ASSIGNED\_DATE, className: '', scope: 'col' },

  { label: import.meta.env.VITE\_CREATED\_DATE, className: '', scope: 'col' },

  { label: import.meta.env.VITE\_REPORT\_STATUS, className: '', scope: 'col' },

  { label: import.meta.env.VITE\_ACTION, className: '', scope: 'col' }

]

pages/auth/auth – used for auth functionality of app.

pages/home/home – used for home page of app.

Routes/AppRoutes – Managing all routes of application.

const AppRoutes = () => {

  return (

    <div>

      <TopNav />

      <Routes>

        <Route path={routePath.ISSUE\_LISTING} element={<Home />} />

        <Route path="/" element={<Auth />} />

        <Route path="/issue-details/:id" element={<IssueDetailsPage />} />

      </Routes>

    </div>

  )

}

### **PYTHON (BACKEND - Success Pilot)**

#### Development Specifications

* **Project:**
  + Success-pilot : This is the main project name.
* **App:**
  + Account : Account app manage user and login related functionality.
  + Dashboard : This app will contain jira issue listing, details and report creation functionality.
* **model:**
  + Model file inside dashboard app containing all table structure for migration.

### BASIC Folder structure

│ celerybeat-schedule.dat

│ celerybeat-schedule.dir

│ manage.py

│ requirements

│

├───account

│ │ admin.py

│ │ apps.py

│ │ backends.py

│ │ models.py

│ │ tests.py

│ │ views.py

│ │ \_\_init\_\_.py

│ ├───migrations

│

├───dashboard

│ │ admin.py

│ │ apps.py

│ │ models.py

│ │ serializers.py

│ │ tests.py

│ │ urls.py

│ │ views.py

│ │ \_\_init\_\_.py

│ ├───migrations

│ │ │ 0001\_initial.py

└───success\_pilot

│ asgi.py

│ celery.py

│ settings.py

│ tasks.py

│ test\_task.py

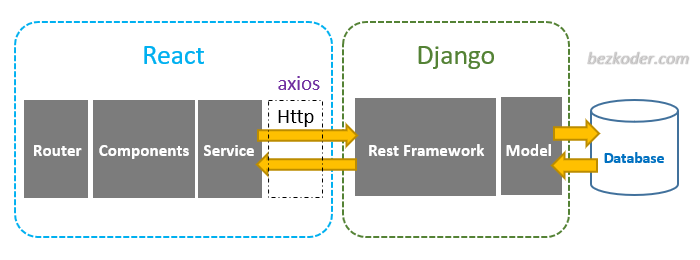
│ urls.py

│ wsgi.py

│ \_\_init\_\_.py

└───\_\_pycache\_

**Django React Architecture**



At the root level of your Django project, find several essential files and directories:

* .manage.py: This is a command-line utility to interact with the Django project.
* Success-pilot/: The inner directory with the same name (success-pilot) as the project will hold the main settings and configurations.
* requirements.txt: It contains a list of project dependencies that can be installed using pip.
* success-pilot/settings.py: As the name suggests, this file houses the settings that configure your Django project. From database configurations to middleware lists, this is where you define how your application functions.
* success-pilot/urls.py: The URL dispatcher — encoded within urls.py—maps URLs to views. This file determines which view is displayed when a specific URL is accessed. It's like a roadmap that navigates users through the intricacies of your application's pages.
* success-pilit/wsgi.py: Short for Web Server Gateway Interface, wsgi.py serves as the entry point for your application when deployed on a production server. It's the bridge connecting your application to the web server, enabling it to handle incoming requests.
* success-pilot/asgi.py: Similar to wsgi.py, asgi.py is the entry point for asynchronous web servers. It stands for Asynchronous Server Gateway Interface and facilitates the handling of asynchronous HTTP requests.
* success-pilot/\_\_init\_\_.py: This seemingly unassuming file holds the magic that transforms a directory into a Python package.

Each app within the “apps” directory (Account, Dashboard) follows a structured layout:

* migrations/: Contains database migration files generated by Django for model changes.
* static/: Holds static files like CSS, JavaScript, and images specific to the app.
* admin.py: Configures the Django admin interface for managing app models.
* apps.py: App configuration and metadata.
* models.py: Defines the app’s data models.
* views.py: Contains the views or controller functions for handling HTTP requests.
* urls.py: Defines the app’s URL patterns.

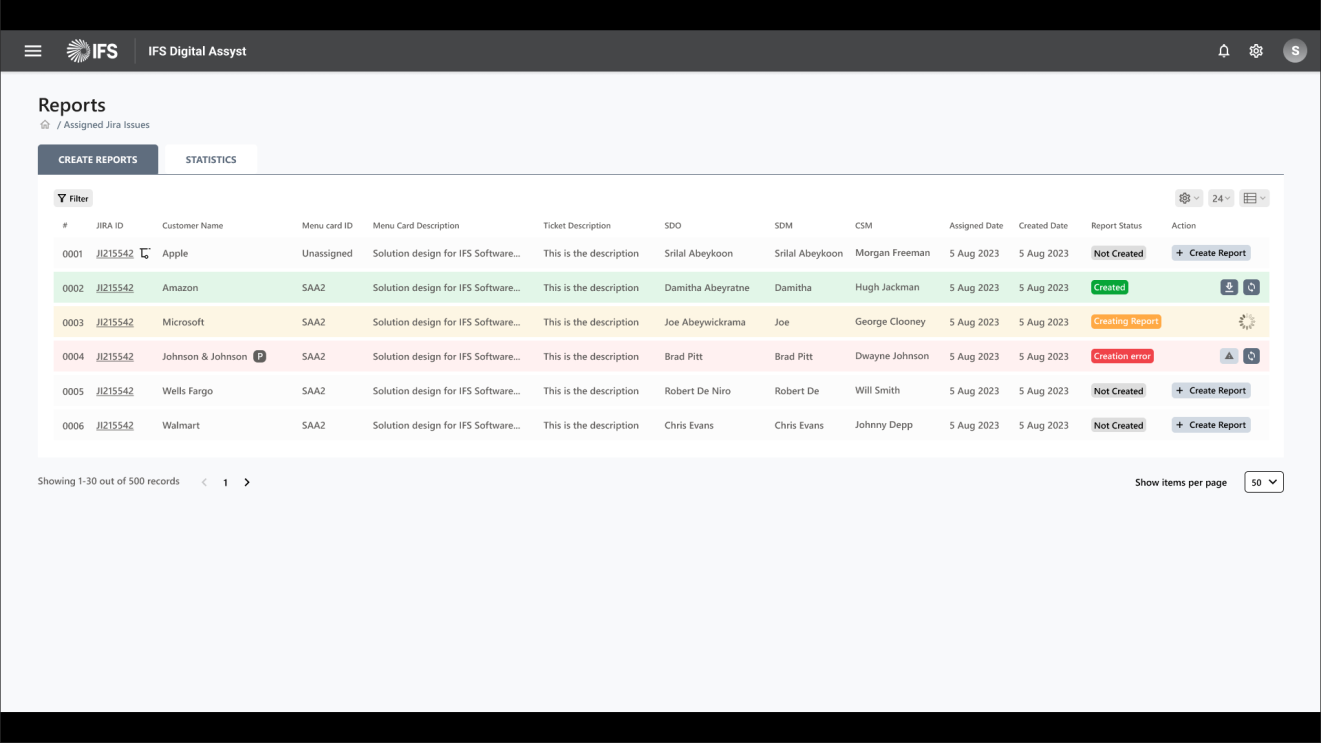
This section explains each functional activity within the report creation module under the report creation module there are multiple screens that help to complete the process, here are a list of all pages:

* **Home Page**
* **List of Assigned Jira Issues**
* **Detail page of Jira issue**
* **Action Items**
* **Schedular for report creation**

**Home Page:** Home will be landing page of the application. Currently login page redirected to issue listing page.Login page authenticated with Microsoft 365 (Azure login).

**List of Assigned Jira Issues:** This page will show all Jira issue (task and sub tasks) assigned to

Login user. After successful login user can see only their assigned Jira issue. Each listing issue having Create Button to create report. After successful created report a download link available for report.



**Backend :**

API : URL/issue\_liting

Request Body : Token [token will contain user details]

Response Sample Body :

[

{

"jira\_id": "SE6000009502-48",

"customer\_name" :"Enterprise Analytics.",

"menu\_id" : "SAA08",

"menu\_desc":"Process Health Check",

"ticket\_desc": "SAA08 Engineers On/Off shift and Record time from arrival to job start",

"sdo":"Andrew Coreless",

"sdm":"Rashadh Azhar",

"csm":"Fredrik Bengtsson",

"assign\_date":"15 Nov 2023",

"created\_date":"16 Nov 2023",

"proect\_status":"2",

"partner":"0",

"subtask":"0",

"subtasks\_list":[{"key": "SE6000009502-59", "summary": "FSM Consultant to write up proposed advisory Final Report"}, {"key": "SE6000009502-60", "summary": "Resource to look into FSM application and document how the solution could be configured in FSM"}],

"customer\_id":"6000009502",

"project\_id": "27280",

"project\_key": "SE6000009502",

"project\_name": "Ericsson AB",

"customer\_email": "sachin.c.gupta@ericsson.com",

"customer\_contact\_no": "705498603",

"customer\_contact": "Anchal Gupta",

"customer\_location": "North & Central Europe"

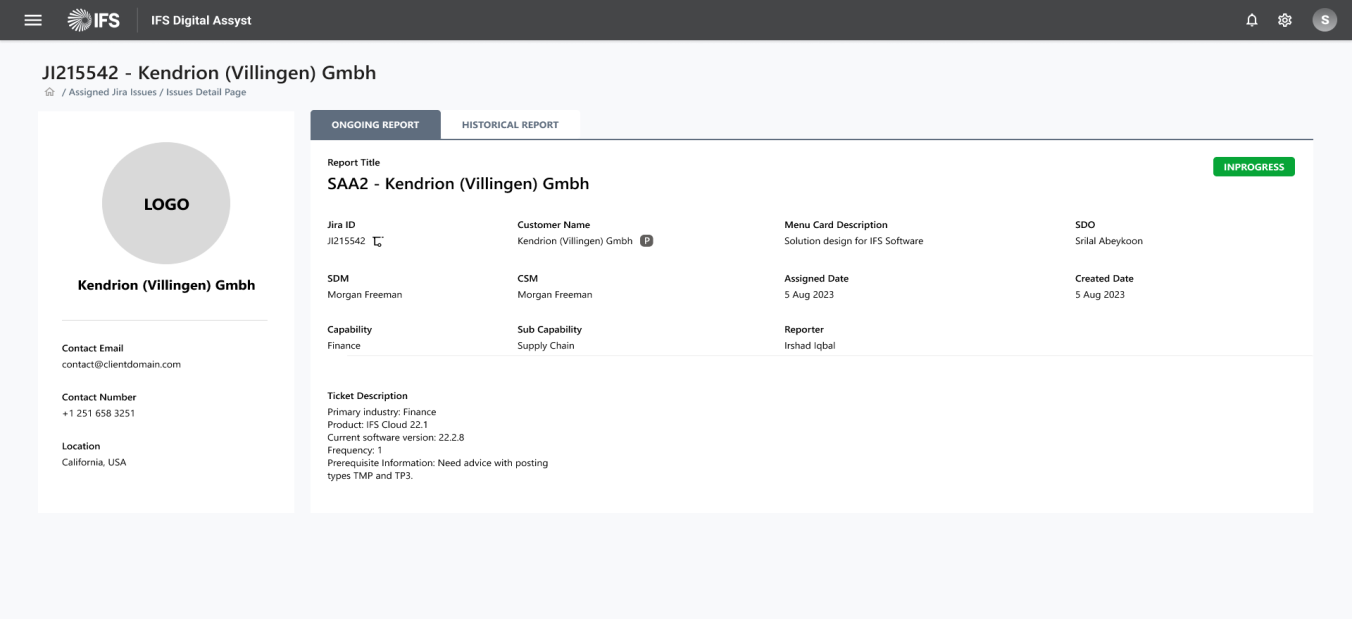
}

]

For each Jira issue field information:

* jira\_id / issue ke **:**Jira key unique identification for each record item will receive from jira.
* menu\_id **: Menu card extracted from activity short name, second last from last item separated from dot(.)**. For example : USX100039.230-EAS.MCEMA11.1
* **Customer Name:** Project/customer name retrieved using the project ID.
* **Menu Card ID:** Name of the menu card item associated with the ticket.
* **Menu Card Description:** Description added by the customer to explain the issue or concern.
* **Ticket Description:** Description provided by the customer during ticket creation.
* **SDO (Service Deliver Officer):** Involved in the delivery process.
* **SDM (Service Deliver Manager):** Involved in the delivery process and client communication.
* **CSM (Customer Service Manager):** Part of services and approval process.
* **Creator:** Person who created the ticket in Snow (Service Now).
* **Assigned Date:** Date the issue was assigned to an expert.
* **Created On:** Date the ticket was created in the Jira system.
* **Report Status:** Indicates the present state of the item, with options like "Not Created," "Report Created and Able to Download," "Creating Report," and "Creation Error."
* **Action:** Buttons for View Details, Create Report, and Delete actions.

**Detail Report Page:**Detail page having all details for that Jira key including client details.



**Report Creation:** Report creation is back ground process that will take require input parameter like Jira Key, Menu Card Customer name, Product, Capability , Sub Capability, Client logo and Creator name to generate report.

**Schedular for report creation:**

**Storage of report:**

### Report Creation

The Report Creation Tool will provide a user-friendly interface with the following key functionalities:

**List of Assigned Jira Issues**

Users can view all Jira issues assigned to them, differentiating between open and closed issues.

**Detail Page of Jira Issue**

On the detail page, users can access comprehensive information about a specific Jira issue, including Action Items and associated details.

**Action Items**

For each Jira issue, the tool will display Action Items with relevant information, including:

* **Item ID:**Jira key unique identification for each record item.
* **Jira ID, Sub Task:** Jira-generated ID containing complete information about the customer and ticket.
* **Customer Name:** Project/customer name retrieved using the project ID.
* **Menu Card ID:** Name of the menu card item associated with the ticket.
* **Menu Card Description:** Description added by the customer to explain the issue or concern.
* **Ticket Description:** Description provided by the customer during ticket creation.
* **SDO (Service Deliver Officer):** Involved in the delivery process.
* **SDM (Service Deliver Manager):** Involved in the delivery process and client communication.
* **CSM (Customer Service Manager):** Part of services and approval process.
* **Creator:** Person who created the ticket in Snow (Service Now).
* **Assigned Date:** Date the issue was assigned to an expert.
* **Created On:** Date the ticket was created in the Jira system.
* **Report Status:** Indicates the present state of the item, with options like "Not Created," "Report Created and Able to Download," "Creating Report," and "Creation Error."
* **Action:** Buttons for View Details, Create Report, and Delete actions.

#### **Additional Features**

* **Subtask Item Description:** Include subtask item description for clarity, specifically for menu card descriptions.
* **Updated/Refreshed Status:** Consider adding a status to identify if the record is "Updated" or "Refreshed," aiding in history logs.
* **History Logs:** Implement a feature to track changes and updates, providing a comprehensive history of actions taken on each record.

##### API End Points

|  |  |  |
| --- | --- | --- |
| End Point name | Request Body | Response |
|  |  |  |
|  |  |  |

*Page Content: Below table will explain each content area of this page:*

|  |  |
| --- | --- |
| Section Name | Description |
| Title | This will be the name of Ticket |
| Breadcrumbs | Name of all parent pages / Page location |
| Action Button | Create report |
| Tickets Information | This section will contain complete information about tickets, like customer info, service menu card etc. you can see the complete information under 4.2.1 section |
| History Chart | This section contains the complete history of past tickets posted by the client. Users can map current issues with older solutions. |

##### 