Update & Submit claim summary

Contents

[1.0 Problem statement 2](#_Toc47546075)

[2.0 Skill Tower develop the project 2](#_Toc47546076)

[3.0 Use Case/Architecture Diagram 3](#_Toc47546077)

[4.0 User Stories 3](#_Toc47546078)

[5.0 Nonfunctional Requirements 5](#_Toc47546079)

[6.0 Expected Deliverables and Implementation Notes 6](#_Toc47546080)

[7.0 Milestone and duration 6](#_Toc47546081)

[8.0 Evaluation rubrics 7](#_Toc47546082)

# 

# 1.0 Problem statement

The purpose of the requirements document is to systematically capture requirements for the project and the system “**Update and Submit claim summary**” to be developed using Front-end framework. Both functional and non-functional requirements have been captured in this document. It also serves as the input for the project scoping.

**About the System**

The client would like to develop an independent application **Update and Submit claim summary** application in order to automate the process of managing the activities.

The following section will cover aspects related to Bank Management System focusing on the front-end features.

1. View Claim summary
2. Update & Submit Claim summary

**Scope of the System**

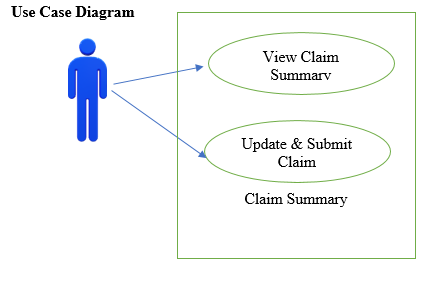
The scope of the system is explained through its modules as follows

* View Claim summary – To view applied Claim details.
* Update & Submit Claim summary – Update & Submit Claim details with required data

# 2.0 Skill Tower develop the project

|  |  |
| --- | --- |
| Tower Name | Skill Names |
| Front end | React  JavaScript  Typescript  React Testing Library |
| Middleware | Nextjs |
| Data Repository | MongoDB |

# 3.0 Use Case/Architecture Diagram



# 4.0 User Stories

|  |  |  |
| --- | --- | --- |
| **User Story #** | **User Story Name** | **User Story** |
| US\_01 | **View Claim Summary Card** | User should be able to see all the active Claims in the form of Claim Summary Cards on the Landing page. In the Claim summary card user should be able to see the following data elements:   1. Claim summary card header 2. Employee ID 3. Employee Name 4. Claim Number 5. Claim Type 6. Claim Programs 7. Claim Start date 8. Claim End date   The employee id will be a request parameter in the launching url.  Individual Claim Summary Card will comprise of the following information:   1. Card title/Header - “**Claim Summary**” 2. Field label – “**Employee Id:**” <Employee Id> display the Employee Id 3. Field label – “**Employee Name:**” <Employee name> display the First and Last Name 4. Field label – “**Claim Number:**” <Claim number> -Need to display the absence number 5. Field label – “**Claim Type:**” <Claim Type> List of Claim types. 6. Field label – “**Claim Program:**” <Claim Program> Claim program to which the employee is enrolled. 7. Field label – “**Claim Start Date:**” <Absence Type> List of Claim types. 8. Field label – “**Claim End Date:**” <Absence Type> List of Claim types.   Link on card “**Update Claim Details”** – On click of this link user will navigate to Update claim info Page |
| US\_02 | **Middleware Claim Summary Data Retrieval** | Create a middleware that will expose an endpoint to fetch a list of “active” Claim details from a Mongo collection comprising of the following information.   1. Employee ID 2. Employee Name 3. Claim Number 4. Claim Type 5. Claim Programs 6. Claim Start date 7. Claim End date   Claim details will be fetched on the basis of Employee ID and the Claim Status |
| US\_03 | **Update Claim- Details Screen** | Trigger - **Update Claim Details link click**  User should be able to see the Claim Summary information from the previously selected Claim Card and be able to edit all fields except Employee ID and Employee Name   1. All details from the Claim Summary Form be displayed in an editable format 2. Fields for Employee ID and Employee Name should be greyed out/disable from edits 3. Next and cancel button should be there in the bottom of page. 4. Next – Navigate to review screen (details in US06) 5. Cancel – On click of cancel User must navigate to Landing Screen 6. Input validation (all are mandatory elements)    * Claim Number – Maximum length of nine digits, only alphanumeric values allowed. Display format XXX-XXX-XXX    * Claim Type – Drop down value – permitted values :   “Submitted”, ”Received”, “Pending”, “More Info Required”, “Denied”, “Rejected”, “Paid”   * + Claim Programs – Maximum length of 20 characters.   + Claim Start date – Date picker to be used (MM/dd/YYYY)   + Claim End date – Date picker to be used (MM/dd/YYYY) |
| US\_04 | **Update Claim- State Management – highlight edited information on Review Page** | Trigger – **Next button click from page1**  The application should be able to store the previously selected information from the landing page, highlight in blue only the edited/updated information on the review screen.  All information displayed on the review screen should be have an edit button next to it clicking upon which the user will be taken back to the previous screen with the focus on the field which is being edited |
| US\_05 | **Update Claim- Claim Review and Submit** | Trigger – **Next button click from page1**  User should be able to see the Submit Claim Summary form in the review page   1. Submit and cancel button should be there in the bottom of page. 2. Submit – Need to Post the form data to Mongo Collection, and need to display success message- submitted successfully 3. Cancel – On click of cancel User must navigate to Summary card |
| US\_06 | **Middleware Claim Date Update** | Create a middleware that will expose an endpoint to update the claim details.   1. Claim Number 2. Claim Type 3. Claim Programs 4. Claim Start date 5. Claim End date |

# 5.0 Nonfunctional Requirements

UI Requirements:

1. The front end should be user-friendly and pleasant
2. The UI screens are designed with any theme color.
3. Any error message or exception displayed to the user should be user-readable (and not technical) and should clearly communicate necessary further actions by the user.
4. All entered values should be validated
5. As focus is on the Front End, static values can be used where ever appropriate
6. Use the input files or mock files to provide static values where ever appropriate

# 6.0 Expected Deliverables and Implementation Notes

Expected Deliverables

* Application Code base
  + .env file for configuring all backed urls
  + .gitignore file for ignoring node\_modules
  + All dependencies expressed using package.json
* Readme document on the complete application
* Testing Report/Screenshots

Developing the requirement using

* React UI framework
  + Styling to be done using css and material-ui
* Implement
  + Forms,
  + Input validations – display appropriate error message when validations are not met
  + Error boundaries – for handling exceptions
* Implement modules using
  + Components
  + Props
  + Hooks
* Avoid using classes rather use functions for developing components
* State Management – using Redux
* React Router – for routing through the screens
* For invoking all backend call
  + Use axios
* Middleware
  + Implemented using Nextjs
* Unit test using
  + React Testing Library
* Coding Standards
  + ESLInt report to be provided

# 7.0 Milestone and duration

|  |  |
| --- | --- |
| Milestone | Topic |
| Milestone -1 | * US\_01 * US\_02 |
| Milestone -2 | * US\_03 * US\_04 |
| Stretch Goal | * US\_05 * US\_06 |

# 8.0 Evaluation rubrics

|  |  |
| --- | --- |
| HTMl5 and CSS3 | * Associate must have used HTML5 pages to render the static and dynamic contents of the application * Associate must have used CSS3 is used for rendering basic styles * Must have used the features of HTML5 and CSS3 to build all the pages needed for the application * UI should be responsive |
| REACT | * Associate must have used Component, props and * Associate must have implemented React Hooks for state and lifecycle methods in function components * Associate must implement lazy loading for react components * Associate must have developed routing using React Router * Associate must have used Forms and Forms validation * Associate must have used React Router * Associate must have implemented React Events * Associate must have used fetch or axios for backend services integration * Associate must have used Typescript-basic features, Configurations and OOPs concepts * Associate must have used Redux Saga features * Associate must have used ES6 and JSX features * Associate must have built application, which is user-friendly and pleasant, and enhance application using React- icons, Internationization. * Associate must have used Error boundaries catch errors during rendering, in lifecycle methods, and in constructors. * Associate must have used ESLint in the MS Visual Code * Associate must have followed coding standards * Associate must have followed best practices * The UI should be responsive (web page adapt to different screen size/resolution like laptop, mobile and tab) |
| JavaScript | * Associate must have used JavaScript Operators, Variables, Databinding, Classes and Objects, Functions, Error Handling * Associate must have Used External and Custom JavaScript * Associate must have used ES6, React JSX |
| Typescript | * Associate must have Used Typescript Operators, Variables, Data types, Directives * Associate must have understanding over typescript concepts |
| Next JS | * Implement any 3 features of Next js - Usage of server-side rendering, Hot Code Reloading, Automatic Code Splitting, Ecosystem Compatibility, Styled-JSX |
| React Testing Library | * Associate must have implemented React Testing Library ,Test cases which covers the functionalities with custom inputs * Associate must have implemented Good test coverage * Associate must run tests and share the screenshots of test results |
| Best practices on React | Below are the criteria’s :   * Use of small functions, each with a single responsibility. This is called the single responsibility principle. This could mean breaking up complex components into many smaller ones. This also will lead to better testability. * Boolean variables, or functions that return a boolean value, should start with “is,” “has” or “should.” * Reusing components across your project Avoid duplication of code – Don’t Repeat Yourself (DRY) * Keep all the CSS styles in a single SCSS file * To name a component after the function that it executes so that it’s easily recognizable * Use capitals for component names * Keep your stateful data-loading logic separate from your rendering stateless logic. * Code should execute as expected and be testable * All files related to any one component should be in a single folder * Code snippets help you to keep up with the best and most recent syntax (There are many snippet libraries that can be used, like, ES7 React, Redux, JS Snippets, etc.) * Write tests for all code |