**Software components, their function and how the components interact with each other**

**CLIENT**

**Survey;**

**ClientContext**

- Handles client wide state management

- Fetch survey from firebase

- Find the last activity where all responses are valid

- Get current activity response. If activity doesn’t have answer generate empty response.

**SurveyContent**

- Component common to survey page only

- Display the questions and pre-populate data based on answer fetched from db if any.

- Update current answer when user selects a response or adds comment

**SummaryContent**

- Component for Summary

- Reads the activity responses from ClientContext and displays this data.

**SurveyComplete**

**-** Component for Survey complete screen.

- Uses the same logic as SummaryContent to create the pdf content.

**Dashboard:**

***Components*:**

* **ClinicianContext:** Provides the current breakpoint (desktop or mobile) to other components.
* **useSnackbarContext:** Provides methods to display snackbar notifications.
* **Router:** Handles navigation between different pages.
* **ClinicianRepo:** Provides functions to fetch data from the "Organization" and "Clinician" collections in Firestore.
* **Firebase:** Provides access to Firestore database.
* **SurveysListDataCompleted, SurveysListDataInProgress, SurveysListDataPending:** Stores survey data for completed, in-progress, and pending surveys, respectively.

***Functions*:**

* **fetchData:**
  + Retrieves information about the clinician's surveys from Firestore.
  + Updates state variables with the retrieved data.
* **addClientButtonClick:**
  + Triggers navigation to the "Add New Client" page.
* **viewAllClickHandler:**
  + Triggers navigation to the "All Questionnaires" page with a query parameter specifying the desired survey status.
* **reloadPageData:**
  + Refetches data from Firestore and updates the UI.

***Component interactions*:**

* ClinicianContext is accessed by the Dashboard component to determine the layout.
* Router is used by Dashboard and SurveyCards components to navigate between pages.
* ClinicianRepo is used by Dashboard and SurveyCards components to fetch data from Firestore.
* Firebase is used by ClinicianRepo to connect to the database.
* SurveysListDataCompleted, SurveysListDataInProgress, and SurveysListDataPending are used by the SurveyCards component to display survey information.
* SnackbarContext is used by the Dashboard component to display error messages.
* Button components trigger functions like addClientButtonClick and viewAllClickHandler.

***Data flow:***

1. Data is fetched from Firestore by the ClinicianRepo and stored in state variables.
2. The SurveyCards component reads the survey data from state variables and renders the UI.
3. User interacts with buttons, triggering functions like addClientButtonClick and viewAllClickHandler.
4. These functions update state variables or navigate to other pages.
5. Changes in state variables cause the UI to be updated accordingly.

***Overall functionality:***

The Clinician Dashboard displays information about the clinician's surveys, including completed, in-progress, and pending surveys. It allows the clinician to add new clients and view all surveys for a specific status.

**All-Questionnaires:**

***Components*:**

* **Container:** Main container for the All Questionnaires page.
* **InnerContainer:** Container for SurveyCards and Pagination components.
* **MainContainerBox:** Container for the main content and the FilterPanel.
* **SearchButton:** Button to open the FilterPanel.
* **StyledBox:** Container for the Tabs component.
* **Tabs:** Tabs to switch between Active and Archived surveys.
* **Button:** Button to reset the filter applied.
* **SurveyCards:** Displays a list of surveys with information like title, created date, client name, and status.
* **Pagination:** Enables navigation through multiple pages of surveys.
* **FilterPanel:** Panel containing filters for client, survey type, survey status, and date.

***Functions*:**

* **useEffect:**
  + Fetches total surveys count and first page of surveys on initial render.
  + Fetches filtered surveys data on filter change.
  + Fetches data for subsequent pages on page change.
* **handleChange:** Handles tab change and updates selected tab.
* **toggleFilterPanelClick:** Opens or closes the FilterPanel.
* **updateFilteredSurveys:** Updates state with filtered surveys data.
* **handleSurveysPageNoClick:** Updates state with the selected page number.

***Component interactions*:**

* **Container:**
  + Contains all other components on the page.
  + Passes props like selected tab and surveys data to relevant components.
* **InnerContainer:**
  + Contains SurveyCards and Pagination components.
  + Receives SurveyCards data from Container.
  + Passes PageNo information to Pagination component.
* **MainContainerBox:**
  + Contains Container and FilterPanel.
  + Provides layout and styles for the page.
* **SearchButton:**
  + Triggers toggleFilterPanelClick function on click.
* **StyledBox:**
  + Provides styling for the Tabs component.
* **Tabs:**
  + Updates selectedTab state on change.
  + Triggers useEffect to fetch data for the selected tab.
* **Button:**
  + Triggers handleResetFilter function on click.
* **SurveyCards:**
  + Displays information about surveys based on props received.
  + Triggers reloadPageData function on click of pagination button.
* **Pagination:**
  + Enables navigation between different pages of surveys.
  + Triggers handleSurveysPageNoClick function on page number click.
* **FilterPanel:**
  + Allows applying filters to surveys data.
  + Triggers updateFilteredSurveys function on filter change.

***Data Flow:***

1. On initial render, total surveys count and first page of surveys are fetched from Firestore.
2. Survey data and selected tab are passed as props to relevant components.
3. User interacts with the FilterPanel and applies filters.
4. Filtered surveys data is fetched from Firestore and state is updated.
5. User interacts with the Pagination component and changes page.
6. Data for the selected page is fetched from Firestore and state is updated.

***Overall functionality:***

The All Questionnaires page allows clinicians to view a list of their clients' surveys, apply filters for specific surveys, and navigate through multiple pages of data.

**My-Clients:**

* **AddNewClient: The component which is used to add a new client to the clinician’s database.**
  + **ClientIdTextField: The input field shown in the component which takes the new unique client id.**
  + **FormControl: Used to display the radio buttons and labels from which clinician can select the first questionnaire type for this new client.**
  + **MobileFullWidthButton: Button used to add the new client by utilizing addNewClient function in clinicianRepo.**
* **AddNewSurveyCard: Used to add new questionnaire to the client.**
  + **FormControl: Used to display the radio buttons and labels from which clinician can select the first questionnaire type for this new client.**
  + **AddSurveyButton: Button used to add the new questionnaire by utilizing addNewSurvey function in clinicianRepo.**
* ClientListCard: Used to display the list of clients for a clinician.
  + StyledSearchBox: Displays a search box through which clients can be searched.
  + StyledClientsList: Displays the list of clients.
  + StyledCardActions: Displays the pagination components which displays the total clients, the number of clients in one page, and buttons to change the client page no.
* ClientListItem: Displays the client id and the date on which that client id was added to the clinician’s database. This component is utilized in ClientListCard component to display in the StyledClientsList.
* SurveysPerClient: Displays New Questionnaire and Filter buttons and the Surveys for the selected client.
  + ButtonsBox: Displays Displays New Questionnaire and Filter buttons and the component which displays selected Filter items.
  + SurveyCards: Displays the list of Questionnaires.
  + Pagination: Displays the pagination components which displays the total questionnaires, the number of questionnaires in one page, and buttons to change the questionnaires page no.
* FilterPanel: Used to display the filter panel components.

**Help:**

* **TabsSelectorBox: The component which is used to display text buttons, which displays different sections as they’re clicked.**
* HowToUseContent: Displays the How to Use section’s components.
  + Accordion: Displays the accordion (list of drop-down headers) which expand and show relevant information when clicked.
* FAQsContent: Displays the Frequently Asked Questions section’s components.
  + Accordion: Displays the accordion (list of drop-down headers) which expand and show relevant information when clicked.

**View Scores:**

***Components*:**

* **Button:** Used for downloading the survey data as CSV.
* **ReportHeader:** Displays information about the survey, such as title, date, and client name.
* **Container, StyledTab, StyledTabs:** Provides the layout and style for the report.
* **ActivityAnalysis:** Displays a pie chart and score for each activity question.
* **getSurveyById:** Fetches survey data from Firestore.
* **parseDataToCsvFormatYoungChild:** Prepares survey data for CSV download.
* **Parser:** Converts JSON data to CSV format.
* **groupByCategory:** Groups survey questions by category.
* **RawScores:** Displays a table of raw scores for all questions.
* **youngChildSurvey:** Provides information about Young Child survey questions.

***Functions*:**

* **downloadCSV:** Triggers the download of the survey data as a CSV file.
* **handleTabValueChange:** Updates the tab value based on user interaction.

***Component interactions:***

* ReportHeader uses survey data from SurveyReport.
* ActivityAnalysis uses survey data from SurveyReport and question ID as props.
* Button triggers downloadCSV function.
* downloadCSV uses parseDataToCsvFormatYoungChild and Parser to generate and download the CSV file.
* CategoryBarChart uses grouped survey data from SurveyReport.
* RawScores uses survey data from SurveyReport.

***Data flow:***

1. Survey data is fetched by getSurveyById and stored in SurveyReport state.
2. ReportHeader reads survey data and displays it.
3. ActivityAnalysis reads survey data and question ID and displays the corresponding pie chart and score.
4. CategoryBarChart reads grouped survey data and displays the bar chart.
5. RawScores reads survey data and displays the table of raw scores.
6. User interaction with the Button triggers the downloadCSV function.
7. downloadCSV uses parseDataToCsvFormatYoungChild and Parser to prepare and download the CSV file.

***Overall functionality:***

The Survey Report displays information and visualizations of a specific survey, including activity analysis, category bar chart, weighted calculation, and raw scores. It also allows the user to download the survey data as a CSV file.

**CLIENT**

**Home:**

***Components*:**

* **HomeContainer:** Main container for the Client Home page.
* **InfoWrapper:** Contains the information section with text and video.
* **ContentWrapper:** Holds the title, description, and introductory video button.
* **YoutubeEmbed:** Displays the introductory video.
* **InfoConfirmWrapper:** Provides information and buttons for confirming participation.
* **ButtonWrapper:** Holds the View Instructions and Start PUFI-2 buttons.
* **Snackbar:** Displays error messages.

***Functions*:**

* **handleStartSurveyClick:** Starts the PUFI-2 survey and returns an error message if any.
* **setHeaderButtonType:** Updates the header button type to "START\_SURVEY".
* **handleViewInstructionsclick:** Navigates to the View Instructions page if a valid survey is present, else displays an error message.
* **handleClose:** Closes the Snackbar.

**Component interactions:**

* **HomeContainer:**
  + Contains all other components on the page.
  + Provides breakpoint information to other components.
* **InfoWrapper:**
  + Displays information and introductory video.
  + Passes breakpoint information to GreyP and InfoConfirmWrapper.
* **ContentWrapper:**
  + Holds the title, description, and button for viewing instructions.
  + Triggers handleViewInstructionsclick on button click.
* **YoutubeEmbed:**
  + Displays the introductory video.
* **InfoConfirmWrapper:**
  + Provides information and buttons for confirming participation.
  + Depends on breakpoint information from InfoWrapper.
* **ButtonWrapper:**
  + Holds the Start PUFI-2 button.
  + Triggers handleStartSurveyClick on button click.
* **Snackbar:**
  + Displays error messages received from handleStartSurveyClick and handleViewInstructionsclick.
  + Can be closed by clicking the Close button.

***Data Flow:***

1. **ClientContext:** Provides organization ID, clinician ID, survey ID, and survey data to ClientHome.
2. **useEffect:** Sets the header button type to "START\_SURVEY".
3. **User interacts with View Instructions button:**
   * If a valid survey exists, navigate to the View Instructions page.
   * Otherwise, display an error message in the Snackbar.
4. **User interacts with Start PUFI-2 button:**
   * Start the PUFI-2 survey.
   * If an error occurs, display it in the Snackbar.
5. **User interacts with the Snackbar:**
   * Close the Snackbar by clicking the Close button.

***Overall functionality:***

The Client Home page provides information about the PUFI-2 questionnaire and allows the user to start the survey or view instructions. It displays error messages in a Snackbar and adapts its layout based on the device breakpoint.

**View Instructions:**

***Components*:**

* **MainContainer:** Main container for the View Instructions page.
* **HomeContainer:** Holds the main content of the page.
* **BeforeStartSurvey:** Displays instructions before starting the survey (shown at instructionId 0).
* **ActivityGuide:** Shows instructions for completing activities (shown at instructionId 1).
* **DifficultyScale:** Allows users to rate the difficulty of activities (shown at instructionId 2).
* **InstructionNavigation:** Provides navigation between instruction sections.

***Functions*:**

* **useState:** Manages state for instructionId and nextButtonDisabled.
* **useContext:** Accesses data from the ClientContext such as organizationId, clinicianId, surveyId, and setDidViewResponseGuide.
* **handleBackClick:** Decreases instructionId if it's not less than 0.
* **handleNextClick:** Increases instructionId if it's less than 2.
* **useEffect:**
  + Redirects to homepage if instructionId is less than 0.
  + Disables the Next button when instructionId is 2 (reached the end).
  + Sets didViewResponseGuide to true when navigating to ActivityGuide (instructionId 1).

***Component interactions:***

* **MainContainer:**
  + Wraps the entire page content.
* **HomeContainer:**
  + Displays the content based on the current instructionId.
  + Passes the navigation props to InstructionNavigation.
* **BeforeStartSurvey:**
  + Shown only when instructionId is 0.
* **ActivityGuide:**
  + Shown only when instructionId is 1.
  + Triggers setDidViewResponseGuide when accessed.
* **DifficultyScale:**
  + Shown only when instructionId is 2.
* **InstructionNavigation:**
  + Displays navigation buttons based on instructionId.
  + Triggers handleBackClick on Back button click.
  + Triggers handleNextClick on Next button click.
  + Renders the Next button as disabled when nextButtonDisabled is true.

***Data Flow:***

1. Initial state sets instructionId to 0 and nextButtonDisabled to false.
2. User interacts with the navigation:
   * Back button: Decreases instructionId and updates the displayed content.

**CLINICIAN**

**Login:**

**LoginContent**

* Display form to login and authenticate using firebase.
* Common information component for clinician login and sign up.

**Components:**

* Styled Container: commonly used container for login page
* Styled Paper: with some elevation shows paper design.
* Styled Heading: heading style for login
* FormBox: designated box styles for form only
* StyledTextFields: specific style for textbox used in form
* StyledButton: button orientation and style

**Component interaction:**

* StyledContainer:
  + Main container wraps all components inside
* StyledPaper:
  + Includes formBox , StyledButton
* FormBox:
  + Wraps StyledTextFields in it
* AccordianCard:
  + Accordinsummary and AccordianDetails are included

**Functions:**

* HandleEmailchange : rects upon changing email and validate
* HandlePAsswordChange : validates password according to firebase rules
* HandleSubmit : calls another function to pass email, password to firebase
* SinginClinicainByEmail : function authenticate clinician
* HandleCloseSnackbar: closes message opened by error events
* HandleForgotPassword: pops up the message with information for clinician to contact their organization.

**Dataflow:**

1. Clincian login by email and password through a form, if credentials matches then user is being directed to dashboard otherwise will show error message and renders same page.

**Register/Create Account:**

**RegisterContent**

* Multipage register clinician form with stepper which takes clinician data
* Every page data is being validated
* After submitting last page, clinician account is created using firebase and redirects to login.
* Common information component is used.

**Components:**

* StyledContainer : commonly used container for login page
* StyledPaper : with some elevation shows paper design.
* StepperBox : stepper component
* FormProvider : uses react’s default useFormContext
* FormBox : styles form fields
* StyledButton : orientation nad styles for buttons
* SnackBar: assigned to every error message
* PersonalDetails: loads form with personal details
* OrganizationDetails: loads form with organization details
* TOS: shows terms of service information

**Component interactions:**

* Styledcontainer:
  + Wraps styled paper components for form and accordion
* StledPaper:
  + Includes stepper, form and buttons in it
* StepperBox:
  + Interacts with stepper activestep as steps increase and decreases changes to page number
* FormProvider:
  + Calls useForm to assign default values of the form

**Functions:**

* PersonalDetails : function renders mui component Personaldetail
* UseFormContext: manages form states
* OrganizationDetails: function renders mui component OrganizationDetails
* TOS : function calls TOS component to show page no 3
* HandleChange : function handles checkbox changed event
* GetStepContent : manages active steps and renders respective components
* UseForm: initialize form with default
* CheckToAndAddCliniacian : checks if TOS was accepted by clinician
* setValidation: calls validation component to validate all fields before moving forward to save clinician
* CheckEmailisUsed : interacts with firebase and makes sure that clinician can use only unique email address.
* CreateClinicianByEmail : implements creating user credentials and createUserWithEmailAndPassword, is a firebase default function
* AddClinicinaDb : adds all fields filled by clinician into firebase
* HandleNext :
  + Checks for next activestep is it is not last step then validates data and direct to next page
  + If last page is found then checks for TOS field
  + If TOS field is fulfilled then moves to storing clinician
  + Clinician added successfully then redirect to login page

**DataFlow :**

1. Clinician will be first presented with register form, step 1 was showing personal detail form, step 2 organzationDetails form , step 3 with Terms Of Service page, with every page clinician data is being stored locally as data can be prepopulate with previous pages, as well as validated.
2. Clinician can not move forward without validating all fields on a single page
3. At last page firebase function is being called to create firebase user, it creates unique id with email, password combination, moreover validated fields will be stored in datastore with unique clinician id
4. Here an empty client object is created which will be populated by add-client functionalities.