calendar.js **Purpose:**

The calendar.js file is a React component that serves as a calendar interface for managing and viewing activities. It includes functionalities for viewing today's events, changing service dates, marking events as complete, and searching for devotees and their activities.

**Key Functionalities:**

1. **State Management**:
   * activities: Holds the list of all activities for the current month.
   * searchResults: Stores the results of a search query.
   * selectedActivity: The activity selected from the calendar for viewing or editing.
   * currentMonth: The current month being viewed in the calendar.
   * accessControl: Stores the user's permissions for interacting with the calendar.
   * searchPerformed: A flag to indicate whether a search operation was performed.
2. **Axios Instance**:
   * A custom axios instance is created with a base URL and authorization header, which is used for making API requests.
3. **Data Fetching**:
   * **fetchAccessControl**: Fetches the user's access control settings based on their user type.
   * **fetchActivities**: Fetches activities for the current month, excluding those with the event name "DONATION".
   * **fetchTodaysActivities**: Fetches today's activities, also excluding "DONATION" events.
4. **Event Handlers**:
   * **handleSearch**: Performs a search for devotees based on the input value and fetches their activities.
   * **handleInputChange**: Handles changes to the search input, triggering a search if the input length is 3 or more characters.
   * **handleDateChange**: Updates the service date of an activity.
   * **handleComplete**: Marks an activity as complete, with a confirmation modal.
   * **handleOk**: Marks the selected activity as complete from the modal.
   * **handleCancel**: Closes the modal without making changes.
   * **handleNavigate**: Updates the current month being viewed in the calendar.
5. **Rendering**:
   * The component renders a layout using Ant Design components and React Big Calendar.
   * **Today's Events in Detail**: Displays detailed information about today's events.
   * **Change Seva Date**: Provides a search interface and table for changing service dates of activities.
   * **Events Calendar**: Displays a calendar view of activities, allowing selection of individual events for more details.

**Code Explanation:**

* **Imports**: The component imports necessary libraries and styles, including React, Ant Design, axios, moment, and React Big Calendar.
* **State Initialization**: Initializes various state variables using React's useState hook.
* **Axios Instance**: Creates a memoized axios instance with authorization headers.
* **Effect Hooks**: Uses useEffect to fetch access control settings on token change, fetch activities on current month change, and fetch today's activities on component mount.
* **Data Fetching Functions**: Defines asynchronous functions to fetch access control settings, activities for the current month, and today's activities. These functions use the axios instance to make HTTP GET requests to the server.
* **Event Handlers**: Defines functions to handle various user interactions, such as searching devotees, changing service dates, marking events as complete, and navigating the calendar.
* **Events Mapping**: Maps activities to calendar events, excluding those with the event name "DONATION".
* **Columns Definition**: Defines columns for the Ant Design Table component used to display search results and today's events.
* **Render Method**: Returns JSX to render the component, including the layout for today's events, the search interface, the calendar view, and the event details modal.

**Conclusion:**

The calendar.js file is a comprehensive React component that integrates various functionalities for managing and viewing calendar activities. It leverages state management, effect hooks, custom axios instance, and Ant Design components to provide a seamless user experience. The component ensures that activities are fetched and displayed accurately, with options for searching, editing, and marking events as complete, all while adhering to the user's access control permissions.Top of Form

Dashboard.js **Purpose:**

The Dashboard.js file is a React component that provides the main layout and navigation for the application. It includes the header, sidebar, and content area, and it manages user authentication, access control, and inactivity timeout for auto sign-out.

**Key Functionalities:**

1. **State Management**:
   * collapsed: Manages the state of the sidebar (collapsed or expanded).
   * username: Stores the username of the logged-in user.
   * headerColor and sidebarColor: Store the colors for the header and sidebar.
   * accessControl: Stores the access control settings for the logged-in user.
2. **Effect Hooks**:
   * **useEffect**: Handles initialization tasks, such as fetching the token, user information, access control settings, and colors from localStorage, and starting the inactivity timeout.
   * **useEffect**: Adds event listeners for user activity to reset the inactivity timeout on user actions like mouse movements, key presses, and clicks.
3. **Inactivity Timeout**:
   * **startInactivityTimeout**: Starts a timeout that triggers auto sign-out after 15 minutes of inactivity.
   * **clearInactivityTimeout**: Clears the inactivity timeout.
   * **handleUserActivity**: Resets the inactivity timeout on user activity.
4. **Fetch Access Control**:
   * **fetchAccessControl**: Fetches access control settings for the logged-in user based on their user type and updates the accessControl state.
5. **User Authentication**:
   * **handleSignOut**: Signs out the user by removing the token from localStorage and navigating to the login page.
6. **Sidebar and Menu**:
   * **toggleCollapsed**: Toggles the state of the sidebar between collapsed and expanded.
   * **menuItems**: Defines the items to be displayed in the sidebar menu based on the user's access control settings.
7. **Breadcrumb Navigation**:
   * **getBreadcrumbItems**: Generates breadcrumb items based on the current path for better navigation.
8. **Routing**:
   * Defines routes for different components like Home, Calendar, Receipts, Reports, SuperAdmin, Settings, and OnlineFormsData, and renders the appropriate component based on the current path.

**Code Explanation:**

* **Imports**: The component imports necessary libraries and components, including React, Ant Design components, icons, routing components, and other application-specific components.
* **State Initialization**: Initializes various state variables using React's useState hook.
* **Effect Hooks**:
  + The first useEffect fetches and sets the token, username, access control settings, and colors from localStorage, and starts the inactivity timeout.
  + The second useEffect adds event listeners for user activity to reset the inactivity timeout and cleans them up on component unmount.
* **Inactivity Timeout Functions**:
  + **startInactivityTimeout**: Clears any existing timeout and sets a new one for 15 minutes.
  + **clearInactivityTimeout**: Clears the current timeout if it exists.
  + **handleUserActivity**: Resets the inactivity timeout whenever user activity is detected.
* **fetchAccessControl Function**: Fetches access control settings from the server based on the user's type and updates the accessControl state.
* **handleSignOut Function**: Removes the token from localStorage and navigates the user to the login page.
* **toggleCollapsed Function**: Toggles the sidebar between collapsed and expanded states.
* **menuItems Array**: Defines the menu items to be displayed in the sidebar, filtered by the user's access control permissions.
* **getBreadcrumbItems Function**: Generates breadcrumb items based on the current path to provide navigation context.
* **Return Statement**: Renders the layout, including the header, sidebar, and content area. It uses Ant Design components for styling and routing components to display the appropriate content based on the current path.

**Conclusion:**

The Dashboard.js file is a central component that manages the overall layout and navigation of the application. It handles user authentication, access control, and inactivity timeout, ensuring a secure and user-friendly interface. The component dynamically adjusts the sidebar menu based on the user's permissions and provides a clear navigation structure with breadcrumbs and routes to various sections of the application.

#### OnlineFormsData.js Purpose:

The OnlineFormsData.js file is a React component that provides an interface for displaying, filtering, and managing data fetched from online forms. It includes functionalities to fetch data from Google Sheets, display it in a table, filter and search the data, update payment status, and delete records.

#### Key Functionalities:

1. **State Management**:
   * data, filteredData: Store the original and filtered data.
   * loading: Indicates whether data is being loaded.
   * amountModalVisible, currentRecord: Manage the visibility of the amount modal and the current record being edited.
   * showAll: Toggles the visibility of all records, including those with 'Paid' status.
   * searchQuery: Stores the current search query.
   * sheetServiceMap: Maps services to their corresponding Excel sheet links.
2. **Data Fetching**:
   * **fetchServiceMap**: Fetches a mapping of services from the server.
   * **fetchExcelSevaData**: Fetches the online forms data and integrates service names.
3. **Data Filtering**:
   * **filterData**: Filters the data based on the search query and showAll flag.
4. **Google Sheets Integration**:
   * **fetchSheetsData**: Fetches the latest data from Google Sheets and updates the state.
5. **CRUD Operations**:
   * **handleDeleteService**: Deletes a service record after confirmation.
   * **handleUpdatePaymentStatus**: Updates the payment status of a record.
   * **handlePaidAtTemple**: Opens the modal to update the payment status to 'Paid'.
6. **Rendering**:
   * The component renders a table with the fetched data, including options to search, filter, update payment status, and delete records.

#### Code Explanation:

* **Imports**: The component imports necessary libraries and components, including React, Ant Design components, axios for API calls, and styles.
* **State Initialization**: Initializes various state variables using React's useState hook.
* **Effect Hooks**: Fetches the service map and online forms data when the component mounts.
* **Data Fetching Functions**:
  + **fetchServiceMap**: Fetches a mapping of services from the server.
    - **fetchExcelSevaData**: Fetches the online forms data and integrates service names.
* **Data Filtering**:
  + **filterData**: Filters the data based on the search query and showAll flag.
* **Google Sheets Integration**:
  + **fetchSheetsData**: Fetches the latest data from Google Sheets and updates the state.
* **CRUD Operations**:
  + **handleDeleteService**: Deletes a service record after confirmation.
  + **handleUpdatePaymentStatus**: Updates the payment status of a record.
  + **handlePaidAtTemple**: Opens the modal to update the payment status to 'Paid'.
* **Render Method**: Returns JSX to render the component, including the layout, table, and modals. The table includes columns for various fields and actions.

#### Conclusion:

The OnlineFormsData.js file is a robust React component that provides functionalities to manage data fetched from online forms. It includes features to fetch data from Google Sheets, display it in a table, filter and search the data, update payment status, and delete records. The component ensures efficient data management and user-friendly interaction with the data through state management, effect hooks, and Ant Design components.

Home.js **Purpose:**

The Home.js file is a React component that provides a comprehensive interface for managing devotees. It includes functionalities to add, edit, delete, and search devotees, as well as add seva (services) for devotees.

**Key Functionalities:**

1. **State Management**:
   * devotees, totalDevotees, currentDevotee, isSevaModalVisible, isModalVisible, services, activeServices, categories, paymentMethods, selectedService, selectedPaymentMethod, familyMembers, accessControl, formKey, formData, emailOptions, isDeleteErrorModalVisible, deleteErrorMessage: Various states to manage the application's data and UI states.
2. **Data Fetching**:
   * **fetchAccessControl**: Fetches access control settings based on the user type.
   * **fetchDevotees**: Fetches the list of devotees from the server.
   * **fetchServices**: Fetches services and categories, and filters active services.
   * **fetchPaymentMethods**: Fetches available payment methods.
3. **CRUD Operations**:
   * **handleAddDevotee**: Prepares the form for adding a new devotee.
   * **handleEditDevotee**: Fetches family members and prepares the form for editing a devotee.
   * **handleDeleteDevotee**: Deletes a devotee after confirming and checking related activities.
   * **handleSeva**: Prepares the form for adding a seva (service) for a devotee.
   * **handleSevaOk**: Handles the addition of a new seva.
   * **handleOk**: Handles the addition or updating of a devotee.
4. **Search and Filtering**:
   * **debounceSearch**: Debounced search function to search devotees based on input.
   * **handleSearchChange**: Updates the search input and triggers the search.
5. **Form Handling**:
   * **handleFamilyChange**, **addFamilyMember**, **removeFamilyMember**: Manage changes in the family members form.
   * **handleEmailChange**: Provides autocomplete suggestions for email input.
6. **Rendering**:
   * The component renders a table with the fetched data, including options to search, filter, update, and delete records.
   * It includes modals for adding/editing devotees and adding seva.

**Code Explanation:**

* **Imports**: The component imports necessary libraries and components, including React, Ant Design components, axios for API calls, and styles.
* **State Initialization**: Initializes various state variables using React's useState hook.
* **Effect Hooks**: Fetches the access control, devotees, services, and payment methods when the component mounts.
* **Data Fetching Functions**:
  + **fetchAccessControl**: Fetches access control settings.
  + **fetchDevotees**: Fetches the list of devotees.
  + **fetchServices**: Fetches services and categories, and filters active services.
  + **fetchPaymentMethods**: Fetches available payment methods.
* **CRUD Operations**:
  + **handleAddDevotee**: Prepares the form for adding a new devotee.
  + **handleEditDevotee**: Fetches family members and prepares the form for editing a devotee.
  + **handleDeleteDevotee**: Deletes a devotee after confirming and checking related activities.
  + **handleSeva**: Prepares the form for adding a seva (service) for a devotee.
  + **handleSevaOk**: Handles the addition of a new seva.
  + **handleOk**: Handles the addition or updating of a devotee.
* **Search and Filtering**:
  + **debounceSearch**: Debounced search function to search devotees based on input.
  + **handleSearchChange**: Updates the search input and triggers the search.
* **Form Handling**:
  + **handleFamilyChange**, **addFamilyMember**, **removeFamilyMember**: Manage changes in the family members form.
  + **handleEmailChange**: Provides autocomplete suggestions for email input.
* **Render Method**: Returns JSX to render the component, including the layout, table, and modals. The table includes columns for various fields and actions.

**Conclusion:**

The Home.js file is a robust React component that provides a comprehensive interface for managing devotees. It includes features to add, edit, delete, and search devotees, as well as add seva (services) for devotees. The component ensures efficient data management and user-friendly interaction with the data through state management, effect hooks, and Ant Design components.

#### Login.js Purpose:

The Login.js file is a React component that provides the user interface for logging in, signing up, and resetting passwords for the Sri Sharadamba Temple, Milpitas website.

#### Key Functionalities:

1. **State Management**:
   * isModalVisible, forgotPasswordEmail, emailError, otp, newPassword, confirmPassword, step, timer, otpSent, isSignupVisible: Various states to manage the UI and form data.
2. **Effect Hooks**:
   * **Timer Management**: Manages the OTP timer using useEffect.
3. **Login Functionality**:
   * **onFinish**: Handles user login by sending a request to the server and storing the token on success.
4. **Forgot Password Functionality**:
   * **handleSendOtp**: Sends an OTP to the user's email.
   * **handleVerifyOtp**: Verifies the OTP entered by the user.
   * **handleResetPassword**: Resets the user's password after OTP verification.
5. **Signup Functionality**:
   * **handleSignup**: Handles user signup by sending a request to the server.
6. **Modal Management**:
   * **showForgotPasswordModal**: Displays the forgot password modal.
   * **handleCancel**: Resets and hides the modal forms.
7. **Utility Functions**:
   * **formatTimer**: Formats the OTP timer for display.
8. **Gear Button Functionality**:
   * **handleGearClick**: Runs gear functions on the server.

#### Code Explanation:

* **Imports**: The component imports necessary libraries and components, including React, Ant Design components, axios for API calls, and styles.
* **State Initialization**: Initializes various state variables using React's useState hook.
* **Effect Hook**: Manages the OTP timer countdown.
* **Form Handling**:
  + **onFinish**: Handles the login form submission.
  + **handleSendOtp**: Sends OTP to the user's email.
  + **handleVerifyOtp**: Verifies the entered OTP.
  + **handleResetPassword**: Resets the user's password.
  + **handleSignup**: Handles the signup form submission.
* **Modal Management**:
  + **showForgotPasswordModal**: Displays the forgot password modal.
  + **handleCancel**: Resets and hides the modal forms.
* **Utility Function**:
  + **formatTimer**: Formats the OTP timer for display.
* **Render Method**: Returns JSX to render the component, including the login form, forgot password modal, and signup modal.

#### Conclusion:

The Login.js file is a comprehensive React component that provides user login, signup, and password reset functionalities. It ensures a smooth user experience with state management, form handling, and modal management using React and Ant Design components. The component also includes an OTP timer and necessary validations to enhance security and usability.

OnlineFormsData.js **Purpose:**

The OnlineFormsData.js file is a React component that provides an interface to manage online forms data. It includes functionalities to fetch, filter, search, update, and delete entries related to sevas (services) performed at the temple.

**Key Functionalities:**

1. **State Management**:
   * data, filteredData, loading, amountModalVisible, currentRecord, showAll, searchQuery, sheetServiceMap: Various states to manage the application's data and UI states.
2. **Data Fetching**:
   * **fetchServiceMap**: Fetches the service map to link services with their IDs.
   * **fetchExcelSevaData**: Fetches the seva data from the server and processes it to include service names.
   * **fetchSheetsData**: Fetches data from Google Sheets.
3. **Filtering and Searching**:
   * **filterData**: Filters the fetched data based on the search query and the showAll flag.
   * **handleToggleChange**: Toggles the showAll flag and filters data accordingly.
   * **handleSearch**: Updates the search query and filters data based on the input.
4. **CRUD Operations**:
   * **handlePaidAtTemple**: Opens a modal to update the payment status of a seva entry.
   * **handleDeleteService**: Deletes a seva entry after confirmation.
   * **handleUpdatePaymentStatus**: Updates the payment status of a seva entry.
5. **Modal Management**:
   * **setAmountModalVisible**: Manages the visibility of the amount modal.
   * **form**: Ant Design form instance to handle form data.
6. **Table Columns**:
   * Defines the columns for the Ant Design table to display the seva data, including actions for updating and deleting entries.

**Code Explanation:**

* **Imports**: Necessary libraries and components including React, Ant Design components, axios for API calls, and styles.
* **State Initialization**: Initializes state variables using React's useState hook.
* **Effect Hooks**: Fetches service map and seva data when the component mounts.
* **Data Fetching Functions**:
  + **fetchServiceMap**: Fetches the service map from the server.
  + **fetchExcelSevaData**: Fetches seva data from the server and processes it.
  + **fetchSheetsData**: Fetches data from Google Sheets.
* **Filtering and Searching**:
  + **filterData**: Filters data based on search query and showAll flag.
  + **handleToggleChange**: Toggles the showAll flag and filters data.
  + **handleSearch**: Updates the search query and filters data.
* **CRUD Operations**:
  + **handlePaidAtTemple**: Opens the amount modal for updating payment status.
  + **handleDeleteService**: Deletes a seva entry after confirmation.
  + **handleUpdatePaymentStatus**: Updates the payment status of a seva entry.
* **Table Columns**: Defines columns for displaying seva data in the Ant Design table.
* **Render Method**: Returns JSX to render the component, including layout, table, and modals.

**Conclusion:**

The OnlineFormsData.js file is a React component that provides a user interface for managing online forms data related to temple services. It includes functionalities for fetching data from the server and Google Sheets, filtering and searching through the data, and performing CRUD operations on the seva entries. The component utilizes Ant Design components for a responsive and user-friendly interface.

Receipts.js **Purpose**:

The Receipts.js file manages receipt data for a temple management system. It includes functionalities for viewing, searching, approving, editing, and managing receipts. Users with appropriate permissions can perform actions such as approving receipts, editing receipt details, downloading, and emailing receipts.

**Key Functionalities**:

1. **Fetching Data**:
   * Retrieves pending, approved, and edited receipts from the server.
   * Fetches access control data based on user type.
   * Retrieves email text for generating PDF receipts.
2. **State Management**:
   * Uses React's useState and useEffect hooks to manage state and side effects.
   * Manages state for receipts, loading status, modal visibility, current activity, and more.
3. **CRUD Operations**:
   * **Approve Receipt**: Allows users to approve receipts, making them available in the approved list.
   * **Edit Receipt**: Enables editing of receipt details and logs the changes.
   * **Search and Filter**: Provides search functionality to filter receipts based on various criteria.
4. **PDF Generation and Emailing**:
   * Uses jsPDF to generate PDF receipts.
   * Allows users to download and print receipts.
   * Provides functionality to email receipts to devotees.
5. **UI Components**:
   * Utilizes Ant Design components such as Table, Button, Modal, Form, and Input for building the UI.
   * Includes pagination and search capabilities for tables displaying receipts.
6. **Access Control**:
   * Checks user permissions for various actions like approving, editing, and emailing receipts based on access control settings fetched from the server.

**Comments and Console Logs**

* Added necessary comments to describe the purpose of each major block of code.
* Included console logs for key CRUD operations to aid in debugging and tracking actions performed on receipts.

This setup ensures the code is clean, maintains functionality, and provides clear documentation for understanding and maintaining the code.

Reports.js  
The Reports component is responsible for generating and displaying reports related to transactions for different services and payment methods within a specified date range. It provides functionalities for filtering, searching, printing, and emailing reports.

1. **State Variables**:
   * services and paymentMethods: Hold the list of services and payment methods fetched from the server.
   * selectedService, selectedPaymentMethod, startDate, endDate: Used for filtering the report data.
   * reportData: Stores the fetched report data.
   * totalDevoteeCount, totalAmount: Hold the calculated totals for the displayed report data.
   * searchQuery, devoteeOptions, selectedDevoteeId, selectedDevoteeName: Handle search functionality for devotees.
   * isEmailModalVisible, currentRecord, email, name: Handle the email modal for sending individual emails.
   * isEmailAllModalVisible, emailAll, emailAllDevoteeName: Handle the email modal for sending emails to all recipients.
   * pageSize: Controls the pagination size for the table.
2. **API Calls**:
   * fetchServices, fetchPaymentMethods: Fetch services and payment methods from the server.
   * generateReport: Fetch report data based on selected filters and set totals.
   * handleSearchChange: Fetch matching devotees based on the search query.
   * handleReEmail: Fetch devotee details for re-emailing.
3. **Event Handlers**:
   * handleServiceChange, handlePaymentMethodChange, handleDevoteeSelect: Update state based on user selections.
   * handleRePrint, handlePrintAll, handlePrintByName: Generate and print PDF reports using jsPDF.
   * handleEmailSend, handleEmailAllSend: Send the generated PDF reports via email.
4. **Rendering**:
   * Date pickers for selecting start and end dates.
   * Dropdowns for selecting services and payment methods.
   * AutoComplete for searching devotees by name, phone, or email.
   * Buttons for printing and emailing reports.
   * Statistic components for displaying total counts and amounts.
   * Table for displaying the filtered report data with actions for printing and emailing individual records.
   * Modals for confirming email actions.

Setting.js  
This file defines a React component for a Settings page in a web application. The page provides various settings and configurations for the application, including password changes, service modifications, theme color adjustments, access rights management, email credentials, and email text customization.

### High-Level Code Explanation:

1. **Imports and Initialization**:
   * The file imports necessary dependencies from React, Ant Design, Axios, JWT decode, and CSS for styling.
   * The Settings component is defined, initializing several state variables using React's useState hook to manage modal visibility, service data, category data, email credentials, theme colors, etc.
2. **Data Fetching**:
   * The useEffect hook is used to fetch services, categories, email text, and general configurations when the component mounts.
   * Functions like fetchServices, fetchCategories, fetchEmailText, and fetchGeneralConfigurations use Axios to make GET requests to the server and update the state with the fetched data.
3. **Service and Category Management**:
   * Functions to handle adding, editing, and saving services and categories, including:
     + handleAddService: Adds a new service.
     + handleAddCategory: Adds a new category.
     + handleServiceSave: Saves the modified services.
     + onTempServiceChange: Handles temporary changes to services.
     + onCategoryActiveChange: Changes the active status of categories.
   * These functions use Axios to send POST and PUT requests to update the server.
4. **Search and Filter**:
   * handleSearch filters services based on a search term.
   * handleSearchSelect selects a service or category from the search results.
   * getDropdownMenu creates a dropdown menu for the search.
5. **Modals and Form Handling**:
   * Various modals are managed using state variables to show and hide them:
     + Password change modal.
     + Service modification modal.
     + New service and category modals.
     + Theme color change modal.
     + Access rights modal.
     + Email credentials modal.
     + Email text edit modal.
   * Forms within modals are managed using Ant Design's Form component.
6. **Theme Colors and Configurations**:
   * Functions to handle changing and resetting theme colors (handleChangeColor, handleResetColors).
   * Functions to manage general configurations like auto-approving users and email confirmation settings.
7. **Access Rights and Email Management**:
   * Functions to fetch and save access rights (fetchAccessControlData, handleAccessRightsSave, handleAccessChange).
   * Functions to fetch and save email credentials (fetchEmailCredentials, handleEmailCredentialsSave, handleEmailChange).
   * Functions to save and reset email text (handleSaveEmailText, handleResetEmailText).
8. **Table and Modal Components**:
   * Ant Design Table components are used to display services and access rights.
   * Ant Design Modal components are used for various settings and configurations, with form inputs for user interaction.

### Comments and Console Logs:

* Necessary comments are added for each main functionality, explaining the purpose of functions and code blocks.
* Console logs are added for debugging and tracking CRUD operations, particularly in data fetching and saving functions.

SuperAdmin.js

This file defines a React component for a Super Admin page in a web application. The page allows a super admin to manage user accounts, including searching, approving, disapproving, deleting users, and changing their access levels.

### High-Level Code Explanation:

1. **Imports and Initialization**:
   * The file imports necessary dependencies from React, Ant Design, Axios, Lodash, JWT decode, and CSS for styling.
   * The SuperAdmin component is defined, initializing several state variables using React's useState hook to manage user data, modal visibility, loading state, etc.
2. **Data Fetching**:
   * The useEffect hook is used to fetch user data and user map when the component mounts.
   * Functions like fetchUsers and fetchUserMap use Axios to make GET requests to the server and update the state with the fetched data.
   * Console logs are added to indicate successful fetching of users and user map.
3. **User Management**:
   * Functions to handle approving, disapproving, deleting, and updating access levels of users:
     + handleAction: Manages the different actions based on the provided action parameter.
     + handleAccessLevelChange: Opens a modal to confirm changing the access level.
     + handleDeleteUser: Opens a modal to confirm deleting a user.
   * These functions use Axios to send PUT and DELETE requests to update the server and refresh the user list.
4. **Search Functionality**:
   * A debounced search function debounceSearch to handle user search input efficiently.
   * handleSearchChange: Updates the search input and triggers the debounced search function.
   * The search functionality ensures that users are filtered and displayed based on the search term.
5. **Table Columns**:
   * The columns array defines the structure of the table, including:
     + User's username, email, access level, approved status, and actions.
     + Access level changes are managed through a Select component.
     + Actions include buttons to approve, disapprove, and delete users.
   * Conditional rendering of the "Super Admin" option based on the logged-in user's username.
6. **Modal Handling**:
   * The state variables isModalVisible, modalContent, and modalAction manage the modal visibility and content.
   * The modal is used to confirm actions like changing access levels or deleting users.
   * The modal's onOk handler triggers the action stored in modalAction.
7. **Layout and Styling**:
   * The component uses Ant Design's Layout and Table components for the overall layout and user table.
   * Custom CSS class site-layout-content is used for styling the content area.
   * The table is made horizontally scrollable on smaller screens using the scroll property.

### Comments and Console Logs:

* Necessary comments are added for each main functionality, explaining the purpose of functions and code blocks.
* Console logs are added for debugging and tracking CRUD operations, particularly in data fetching and action handling functions.

TodaysEvents.js

This file defines a React component for displaying today's events and Panchanga (Hindu calendar details) for a selected date. It allows users to navigate between dates and view specific events and Panchanga details for the selected date.

**High-Level Code Explanation:**

1. **Imports and Initialization**:
   * The file imports necessary dependencies from React, Ant Design, Axios, Moment, and a CSS file for styling.
   * The TodaysEvents component is defined, initializing state variables using React's useState hook to manage events, selected date, and Panchanga details.
2. **Data Fetching**:
   * The useEffect hook is used to fetch events and Panchanga details when the component mounts and whenever the selected date changes.
   * Functions fetchEvents and fetchPanchanga use Axios to make GET requests to the server and update the state with the fetched data.
   * Console logs are added to indicate successful fetching of events and Panchanga.
3. **Date Navigation**:
   * Functions handlePreviousDay and handleNextDay adjust the selectedDate state to navigate between previous and next days.
4. **Rendering Event Cards**:
   * The renderEventCard function generates a card for each service, excluding "DONATION".
   * It maps through the activities of each service to display details like name, Gotra, Star, and family members.
5. **Rendering Panchanga Details**:
   * The renderPanchanga function generates a card displaying Panchanga details like Tithi, Weekday, Nakshatra, Yoga, and Karana.
6. **Layout and Styling**:
   * The component uses Ant Design's Layout, Row, Col, Card, Typography, and Button components for the overall layout.
   * Custom CSS class todays-events-container is used for styling the content area.
   * The date navigation buttons and date title are displayed in a flex container for center alignment.

**Comments and Console Logs:**

* Necessary comments are added for each main functionality, explaining the purpose of functions and code blocks.
* Console logs are added for debugging and tracking CRUD operations, particularly in data fetching functions.

TV.js

This file defines a React component for displaying a TV-like view of temple events, Panchanga (Hindu calendar details), and Sevas (services) for the current day. It includes a slideshow of images and a detailed view of the activities.

**High-Level Code Explanation:**

1. **Imports and Initialization**:
   * The file imports necessary dependencies from React, Axios, Moment, and a CSS module for styling.
   * The TV component is defined, initializing state variables using React's useState hook to manage date and time, images, events, Panchanga details, and activities.
2. **Date and Time Management**:
   * The useEffect hook is used to update the current date and time every second.
3. **Data Fetching**:
   * The useEffect hook is used to fetch events, Panchanga details, images, and activities from the server when the component mounts.
   * Functions fetchEvents, fetchPanchanga, fetchImages, and fetchActivities use Axios to make GET requests to the server and update the state with the fetched data.
   * Console logs are added to indicate successful fetching of data.
4. **Slideshow and Activity Display**:
   * The useEffect hook is used to manage the slideshow and activity display, switching between them at intervals.
   * Progress and image index are updated at regular intervals to control the slideshow.
5. **Utility Functions**:
   * formatDate, formatDayDate, isToday, and formatDay functions are used to format dates for display.
   * resetTime function resets the time to midnight for date comparison.
   * groupActivitiesByService groups activities by service for easier rendering.
6. **Service Image Mapping**:
   * The getServiceImage function maps service IDs to corresponding images.
7. **Rendering Components**:
   * The component uses JSX to render different sections of the UI, including the welcome text, temple events, Panchanga details, current date and time, temple timings, slideshow, and activities.
   * Conditional rendering is used to switch between slideshow and activities view based on the current mode and presence of activities.

**Comments and Console Logs:**

* Necessary comments are added for each main functionality, explaining the purpose of functions and code blocks.
* Console logs are added for debugging and tracking data fetching operations, particularly in data fetching functions.

TV1.js  
This file defines a React component for displaying a TV-like view of temple events, Sevas (services) for the current day, and a slideshow of images. The component toggles between showing images and tables of events and activities.

### High-Level Code Explanation:

1. **Imports and Initialization**:
   * The file imports necessary dependencies from React, Axios, and a CSS module for styling.
   * Images for different services are imported for display purposes.
   * The TV1 component is defined, initializing state variables using React's useState hook to manage date and time, images, events, and activities.
2. **Date and Time Management**:
   * The useEffect hook is used to update the current date and time every second.
3. **Data Fetching**:
   * The useEffect hook is used to fetch events, images, and activities from the server when the component mounts.
   * Functions fetchEvents, fetchImages, and fetchActivities use Axios to make GET requests to the server and update the state with the fetched data.
   * Console logs are added to indicate successful fetching of data and to track errors.
4. **Content Display Management**:
   * The useEffect hook is used to manage the slideshow and content display, switching between them at intervals.
   * Progress and current index are updated at regular intervals to control the slideshow and toggle between image and table views.
5. **Utility Functions**:
   * formatDate, formatDayDate, isToday, and resetTime functions are used to format dates for display.
   * groupActivitiesByService groups activities by service for easier rendering.
6. **Service Image Mapping**:
   * The getServiceImage function maps service IDs to corresponding images.
7. **Rendering Components**:
   * The component uses JSX to render different sections of the UI, including the slideshow, upcoming temple events, and today's Sevas.
   * Conditional rendering is used to switch between the slideshow and table views based on the current state.

### Comments and Console Logs:

* Necessary comments are added for each main functionality, explaining the purpose of functions and code blocks.
* Console logs are added for debugging and tracking data fetching operations, particularly in data fetching functions.

Index.js The code defines and sets up several Sequelize models for an application, including associations between the models. It covers configuration, model definitions, associations, and synchronization with the database.

### High-Level Code Explanation

1. **Imports and Configuration**:
   * Imports necessary libraries (Sequelize, uuid) and environment configuration.
   * Sets up database configuration for different environments (development, test, production).
2. **Database Initialization**:
   * Initializes a new Sequelize instance with database configurations.
3. **Model Definitions**:
   * Defines various models using sequelize.define(), specifying fields and their properties.
   * Models include User, Devotee, Family, Service, ServiceCategory, ModeOfPayment, Activity, Receipt, AccessControl, EmailCredential, GeneralConfigurations, EditedReceipts, and ExcelSevaData.
4. **Associations**:
   * Sets up relationships between models using Sequelize's association methods (hasMany, belongsTo).
5. **Database Synchronization**:
   * Synchronizes the defined models with the database, creating tables if they do not exist.
   * Logs success or error messages based on the outcome of the synchronization process.

### Comments and Console Logs

* Necessary comments are added for each main functionality, explaining the purpose of sections and model definitions.
* Console logs are added for debugging and tracking database synchronization operations, particularly when syncing models with the database.

server.js

The documentation for the server.js file, explaining the purpose and functionality of each section, is as follows:

### Dependencies and Configurations

* **Required Modules**: The script imports necessary modules such as express, body-parser, cors, bcrypt, jsonwebtoken, sequelize, moment, multer, nodemailer, cron, and axios.
* **Sequelize Models**: Various models from Sequelize are imported, which represent different entities in the database.
* **Google APIs**: Google Sheets and Drive API configurations are set up using service account credentials.
* **Application and Port**: An Express application is created, and the server listens on port 5001.

### Middleware

* **Body Parser**: Parses incoming request bodies in a middleware before handlers.
* **CORS**: Enables Cross-Origin Resource Sharing for all routes.

### Authentication Middleware

* **authenticateToken**: Middleware function that verifies the JWT token provided in the request header for protected routes.

### Routes

#### User Management

* **Get All Users**: GET /users - Fetches all users with basic details.
* **Get User by ID**: GET /user/:userid - Fetches a specific user by userid.
* **Signup**: POST /signup - Registers a new user with username, email, and password.
* **Login**: POST /login - Authenticates a user using username or email and password.
* **Change Password**: POST /change-password - Allows authenticated users to change their password.
* **Forgot Password**: POST /forgot-password - Initiates the password reset process by generating an OTP and sending it via email.
* **Verify OTP**: POST /verify-otp - Verifies the OTP sent for password reset.
* **Reset Password**: POST /reset-password - Resets the password using the verified OTP.
* **Invalidate Token**: POST /invalidate-token - Invalidates the password reset token.

#### Devotee Management

* **Get All Devotees**: GET /devotees - Fetches all devotees.
* **Get Devotee by ID**: GET /devotee/:id - Fetches a specific devotee by id.
* **Search Devotees**: GET /devotees/search - Searches devotees by various parameters.
* **Create Devotee**: POST /devotees - Creates a new devotee with optional family details.
* **Update Devotee**: PUT /devotees/:id - Updates an existing devotee's details.
* **Delete Devotee**: DELETE /devotees/:id - Deletes a devotee by id.

#### Service Management

* **Get All Services**: GET /services - Fetches all services.
* **Create Service**: POST /services - Creates a new service.
* **Update Services**: PUT /services - Updates multiple services.
* **Get Categories**: GET /categories - Fetches all service categories.
* **Create Category**: POST /categories - Creates a new service category.
* **Update Category**: PUT /categories/:id - Updates a service category.
* **Delete Category**: DELETE /categories/:id - Deletes a service category.

#### Activity Management

* **Get All Activities**: GET /activities - Fetches all activities.
* **Create Activity**: POST /activities - Creates a new activity.
* **Update Activity**: PUT /activities/:id - Updates an existing activity.
* **Delete Activity**: DELETE /activities/:id - Deletes an activity by id.

#### Receipt Management

* **Get Pending Receipts**: GET /receipts/pending - Fetches all pending receipts.
* **Get Approved Receipts**: GET /receipts/approved - Fetches all approved receipts.
* **Approve Receipt**: POST /receipts/approve - Approves a receipt.
* **Send Receipt Email**: POST /send-receipt-email - Sends a receipt email with an attachment.

#### General Configurations

* **Get Configurations**: GET /general-configurations - Fetches general configurations.
* **Update Configurations**: PUT /general-configurations - Updates general configurations.

#### Access Control

* **Get Access Control by UserType**: GET /access-control/:userType - Fetches access control data for a specific user type.
* **Get All Access Control Data**: GET /access-control - Fetches all access control data.
* **Update Access Control**: PUT /access-control - Updates access control data.

#### Email Credentials

* **Get Email Credentials**: GET /email-credentials - Fetches email credentials.
* **Update Email Credentials**: PUT /email-credentials - Updates email credentials.

#### Event and Panchanga Data

* **Get Events**: GET /api/events - Fetches cached events data.
* **Get Panchanga**: GET /api/panchanga - Fetches cached panchanga data.
* **Get Today's Activities**: GET /api/today-activities - Fetches activities for today.
* **Get Images**: GET /api/images - Fetches image URLs for the slideshow.
* **Get Image by Filename**: GET /api/image/:filename - Fetches a specific image by filename.

### Google Sheets and Drive Integrations

* **fetchEvents**: Fetches events from Google Sheets.
* **fetchPanchanga**: Fetches Panchanga data from Google Sheets.
* **fetchImages**: Fetches images from Google Drive.

### Background Tasks

* **cron.schedule**: Schedules tasks to run periodically to fetch events, panchanga, and images.

### Server Initialization

* **sequelize.sync**: Synchronizes the database and creates initial super user and email credentials if not present.
* **app.listen**: Starts the Express server on the specified port.

This documentation outlines the structure and functionality of the server.js file, providing a comprehensive overview of the API endpoints, background tasks, and integrations with external services.