Index

changeNavigationHighlightSummary
changeNavigationPrivacyPolicy
changePriorityColor
checkAssignContact
checkCards
checkErrors
checkForDuplicateEmail
checkIfEmailExists
checkIfUserIsAddedAsContact
checkInputValue
checkPasswordStrength
checkPrivacyPolicy
clearAddContactForm
clearPasswordError
clearSubtaskInputField
clickedButtonEmailColors
clickedButtons
clickedLegalPart
closeAssignToDropdown
closeCard
closeContactInfo
closeDropdowenTask
closeDropdown
closeTaskCategoryDropdown
createCardObject
createCategories
createContact
createNewContactDataSet
createTask
createUserAsContact
currentDraggedElement
deleteContact
deleteCreatedSubtask
deleteTask
deleteUneditTask
displayClosestDueDate
doNotClose
doneCardUpdate
doneNumber
drop
dropMobile
dropdowenTask
dropdownHelp

```
dropdownLegalNotice
dropdownPrivacyPolicy
editContactDeleteAndSaveButtonLayoutHTMLTemplate
editCreatedSubtask
editTask
editTaskDone
emptyArrays
errorMessageIfEmptyCategory
errorMessageIfEmptyDueDate
errorMessageIfEmptyTitle
feedbackCardUpdate
feedbackNumber
formatContactName
formatDueDate
getCategoryColor
getContactData
getContactRelatedInfo
getIndexByNameSurname
getOldestOverdueDate
getPriorityImagePath
getQueryParam
getSelectedContact
getTaskData
giveId
goBack
greetUser
greetUserMobile
guestLogin
handleContactAssignment
handleRememberMeChange
hideAddContactCard
hideContactEditDeleteMenu
highlightCreatedContact
highlightErrorMessage
includeAddTask
includeHTML
init
initContacts
insertAnimation
isAssignedEdit
legalNoticeHiglite
loadAddTasks
loadBoard
```

```
loadContacts
loadRememberedPassword
loadRemoteContactsOfLoggedInUser
loadTasks
loadUsers
login
logout
lowPriorityButtonStylingWhenClicked
matchingCategoryCheck
mediumPriorityButtonStylingWhenClicked
mobileGreeting
navigateTo
navigationClick
navigationClickImg
newContactDataSetForArray
openAssignToDropdown
openContactInfo
openContactInfoHTMLTemplate
openCreatedSubtaskBox
openDropdown
openTaskCategoryDropdown
outputDeadlineText
outputDueDate
priorityEdit
priorityImgCheck
privacyPolicyHighlight
progressCardUpdate
progressNumber
progressbarCompetedRate
redesignAddContactCardToEditContactCard
rememberPassword
removeClickedLegalPart
removeNavHighlightLegalPartOnDropdown
removeNavHighlightOnDropdown
removeNavigationClick
renderAddContactLayout
renderAllContacts
renderContactCategory
renderContactCategoryAndEachContact
renderContactList
renderEachContact
renderFilteredContacts
resetAddTaskForm
```

```
resetContactButtonColor
resetContactButtonEmailColor
resetCreateTaskFormInputs
resetLastClickedContactButtonColor
resetLastClickedContactButtonEmailColor
resetNavigationItems
resetOtherPriorityButtonStyles
saveEditSubtaskInput
saveNewUser
saveSubtaskInput
scrollDown
scrollToAnchor
search
searchContactToAssign
selectTaskCategory
separateCards
setButtonColorAsClicked
setEmailColorAsClicked
setGuestLogin
setUserLogin
showAddContactCard
showAvatarsOfSelectedContacts
showContactCreatedPopUp
showContactEditDeleteMenu
showCurrentContactDetails
showEditAndDeleteMenuOnMobile
showEditContact
showTaskCreatedPopUp
showTaskCreatedPopUpBoard
sortByFirstName
sortUpcomingCards
startAnimation
startDragging
subtaskCompleted
subtaskEdit
subtaskNotCompleted
successfulLogin
successfulSignup
summaryLoad
summaryLoadNumbers
templateOkBtn
todoCardUpdate
todoNumber
```

toggleAssignToDropdown
toggleCheckbox
toggleContactButtonColor
toggleContactButtonEmailColor
toggleIndex
togglePassword
togglePrivacyPolicyCheckbox
toggleRememberMeCheckbox
toggleSelectTaskCategoryDropdown
uncheckAssignContact
unclickCreatedContact
updateCards
updateContact
updateDueDateContainers
updateUserContactsInRemote
updateUserContactsInRemoteAfterRegistration
urgentButtonColor
urgentNumber
urgentPriorityButtonStylingWhenClicked
validateConfirmedPassword
validateEmailAddress
validateFormFields
validateTaskData
wrongPasswordMessage

Methods

`boardPlace`

This function conditionally opens the "Add Task" functionality based on screen size. - For small screens (less than 800px width), it calls the `loadAddTasks` function to handle adding tasks in a compact format. - For larger screens, it calls the `boardPopupAddTask` function to display a popup for adding tasks).

Source:

addTask_part1.js, line 53

`cards`

Array to implement the Task cards !!

Source:

board_part1.js, line 4

`clickedButtonEmailColors :Array.<string>`

This variable stores an array of IDs for email elements within clicked contacts. Used to keep track of clicked contact email colors for styling purposes.

Type:

- Array.<string>

Source:

contacts_part2.js, line 109

`clickedButtons :Array.<string>`

This variable stores an array of IDs for buttons that have been clicked on the contact list. Used to keep track of clicked contacts for styling purposes.

Type:

- Array.<string>

Source:

contacts_part2.js, line 102

`currentDraggedElement :number`

A global variable to store the ID of the currently dragged card element.

Type:

- number

Source:

board_part1.js, line 218

`toggleIndex :number`

This variable stores the index of the currently highlighted contact (for screens wider than 800px).
Used to keep track of which contact is visually selected.

Type:

- number

Source:

contacts_part2.js, line 273

`CreatedPopUpOptions() → {void}`

Selects and displays the appropriate popup based on element availability.

Source:

addTask_part2.js, line 308

Returns:

(nothing returned)
Type

void

`EditTemplate() → {string}`

Generates the HTML structure for a large board card popup.

Source:

templates.js, line 280

Returns:

The HTML string representing the board card popup structure.
Type

string

`NoMatchFound(hasMatch, query)`

Handles no search results: shows message if no match, clears if query is empty or has matches.

Parameters:

Name	Type	Description
`hasMatch`	boolean	Flag indicating if any matches were found.
`query`	string	The search query string.

Source:

board_part2.js, line 80

`SubtaskStatus(done, i, id)`

Updates the visual state (image) and internal completion status of a subtask. Triggers UI updates and opens the big card modal for the associated card.

Parameters:

Name	Type	Description
`done`	boolean	Indicates whether the subtask is marked as completed.
`i`	number	The index of the subtask within the card's subtasks array.
`id`	number	The ID of the card containing the subtask.

Source:

board_part1.js, line 389

`TaskCadBigTemplate(card) → {string}`

Generates the HTML structure for the big card modal content based on a card object.

Parameters:

Name	Type	Description
`card`	object	The card object containing details to populate the template.

Source:

templates.js, line 156

Returns:

The HTML string representing the big card modal content.
Type

string

`TaskTextInEdit(id)`

Pre-fills edit form fields with data from the task object based on its ID.

Parameters:

Name	Type	Description
`id`	string	The ID of the task to edit.

Source:

board_part2.js, line 114

`TemplateGreetMobile()`

Generates the HTML structure for a mobile greeting element. Inserts the greeting content into the element with ID "content".

Source:

templates.js, line 313

`TemplateSubtaskProgressbar(card)`

check if ther are subtask to put on the card for the todo card.

Parameters:

Name	Type	Description
`card`	*	the arry of the card for the needet task

Source:

templates.js, line 134

Returns:

the Subtask section on the bord task card.

(async) `Templates(template)`

Loads the specified template file and includes its content in the designated container.

Parameters:

Name	Type	Description
`template`	string	The name of the template file to load (without the extension).

Source:

script.js, line 166

(async) `UpdateTaskInRemote()` → {Promise.<void>}

Saves the current tasks array to local storage.

Source:

board_part1.js, line 32

Returns:

A promise that resolves after saving the tasks.

Type

Promise.<void>

`UserInitals()`

Displays user initials based on the provided user name on the board

Source:

login.js, line 233

`addContactButtonsCancelAndCreateButtonsHTMLTemplate() → {string}`

Generates the HTML template for the cancel and create buttons used in the "Add Contact" card.

Source:

contacts_part1.js, line 258

Returns:

The HTML string containing the button elements.
Type

string

`addTaskInit()`

Initializes functionalities related to adding tasks. This function likely performs actions to prepare the "add task" functionality. It calls `changePriorityColor` for potential default priority color setting.

Source:

addTask_part1.js, line 21

`addTaskToBoard(newCard)`

This function adds the provided new task object (presumably containing task details) to the `cards` array. The `cards` array likely represents the collection of tasks displayed on the board. Additionally, it calls the `UpdateTaskInRemote` function (assumed to be defined elsewhere) to potentially update the task data remotely.

Parameters:

Name	Type	Description
`newCard`	Object	The new task object to be added to the board.

Source:

addTask_part2.js, line 405

`(async) addUser() → {Promise.<void>}`

Adds a new user after form validation and redirects to login page on success.

Source:

signup.js, line 153

Returns:

Type

Promise.<void>

`allowDrop(ev)`

Allows dropping an element onto a designated target.

Parameters:

Name	Type	Description
`ev`	DragEvent	The drag event object.

Source:

board_part1.js, line 232

`assignedInitals(card)`

Initializes the assigned user initials display for a given card.

Parameters:

Name	Type	Description
card	object	A card object containing an `assigned` array with user information. - The card object is expected to have an `assigned` property which is an array of objects. - Each object in the `assigned` array should have a `name` property (string) and a `color` property (string).

Source:

board_part1.js, line 183

assingContact(i)

This function handles assigning or unassigning a contact based on the provided index.

Parameters:

Name	Type	Description
i	number	The index of the contact in the list.

Source:

addTask_part1.js, line 307

bigCard(id)

Opens a big card modal for a specific card based on its ID.

Parameters:

Name	Type	Description
id	number	The ID of the card to display.

Source:

board_part1.js, line 286

bigCardAssigned(card)

Initializes the assigned user list display for the big card view based on the provided card data. If the card has no assigned users, hides the container.

Parameters:

Name	Type	Description
`card`	object	A card object containing an \`assigned\` array with user information. - The card object is expected to have an \`assigned\` property which is an array of objects. - Each object in the \`assigned\` array should have a \`name\` property (string) and a \`color\` property (string).

Source:

board_part1.js, line 315

`bigCardAssignedTemplate(user, initials) → {string}`

Generates the HTML template for a single assigned user within the big card view.

Parameters:

Name	Type	Description
`user`	object	A user object containing a \`name\` property (string) and a \`color\` property (string).
`initials`	string	The user's initials (uppercase) generated from their name.

Source:

templates.js, line 223

Returns:

The HTML template string representing a single assigned user.

Type

string

`bigCardSubtaskTemplate(taskText, img, done, i, id) → {string}`

Generates the HTML template for a single subtask within the big card modal.

Parameters:

Name	Type	Description
`taskText`	string	The text content of the subtask.
`img`	string	The path to the image representing the subtask state.
`done`	boolean	Indicates whether the subtask is marked as completed.
`i`	number	The index of the subtask within the card's subtasks array.
`id`	number	The ID of the card containing the subtask.

Source:

templates.js, line 242

Returns:

The HTML string representing the subtask template.
Type

string

`bigCardSubtasks(card)`

Populates the subtasks section of the big card modal with individual subtask details.

Parameters:

Name	Type	Description
`card`	object	The card object containing the subtasks array.

Source:

board_part1.js, line 362

`bigCardSubtasksCheck(card)`

Checks for subtasks and conditionally hides the subtasks area of the big card modal.

Parameters:

Name	Type	Description
`card`	object	The card object containing the subtasks array.

Source:

board_part1.js, line 350

`(async) boardPopupAddTask()`

Opens the modal for adding a new task to the board. Fetches the add task template using the 'include-AddTask' attribute and injects it into the modal content.

Source:

board_part2.js, line 40

`boardPopupAddTaskWindow() → {string}`

Generates the HTML template for the "Add Task" popup window.

Source:

templates.js, line 255

Returns:

The HTML string representing the popup window structure.
Type

string

`boardTaskNumber()`

Calculates and displays the total number of tasks in the board using the 'cards' array length. Performs error handling if the container element with ID 'bord-tasks-number' is not found.

Source:

summary.js, line 141

`buildTemplateForArrayInput()`

This function builds a new task object with the provided details. It constructs the object with properties like `id`, `place`, `category` (including name and color), `title`, `description`, `dueDate`, `subtasks`, `assigned contacts`, and `priority` (including urgency and image path).

Source:

addTask_part2.js, line 378

`capitalizeFirstLetter(name) → {string}`

This function capitalizes the first letter of a string.

Parameters:

Name	Type	Description
`name`	string	The string to be capitalized.

Source:

contacts_part1.js, line 383

Returns:

The string with the first letter capitalized.
Type

string

`cardTemplate(card) → {string}`

Generates the HTML structure for a single card on the board.

Parameters:

Name	Type	Description
`card`	object	The card object containing details to populate the template.

Source:

templates.js, line 105

Returns:

The HTML string representing a card on the board.
Type

string

`changeBackCheckBoxStyle(i)`

This function changes the source attribute of a checkbox element with a dynamic ID constructed using `assignContactCheckBox(i)`. The new source points to the "unchecked" checkbox image.

Parameters:

Name	Type	Description
`i`	number	The index of the contact in the list.

Source:

addTask_part1.js, line 256

`changeCheckBoxStyle(i)`

This function changes the source attribute of a checkbox element with a dynamic ID constructed using `assignContactCheckBox(i)`. The new source points to the "checked" checkbox image.

Parameters:

Name	Type	Description
`i`	number	The index of the contact in the list.

Source:

addTask_part1.js, line 246

`changeContactButtonColorAsClicked(index)`

This function handles clicks on contact buttons in the contact list. It checks the screen size and performs different actions based on the width. - For screens wider than 800px: - Unclicks any previously clicked contact. - Gets the ID of the clicked button and its DOM element. - Toggles the clicked button's color and style. - Changes the email color of the clicked contact to white. - For screens smaller than 800px: - Opens the contact info for the clicked contact.

Parameters:

Name	Type	Description
`index`	number	The index of the clicked contact in the `contacts` array.

Source:

contacts_part2.js, line 123

`changeContactButtonEmailColorToWhite(index)`

This function changes the email color of a clicked contact to white (for screens wider than 800px). It retrieves the email element's ID and DOM element based on the clicked contact index. Then, it calls the `toggleContactButtonEmailColor` function to handle the color change logic.

Parameters:

Name	Type	Description
`index`	number	The index of the clicked contact in the `contacts` array.

Source:

contacts_part2.js, line 206

`changeLockIcon(inputElement)`

Changes the visibility icon of the password input field.

Parameters:

Name	Type	Description
`inputElement`	HTMLElement	The password input field element.

Source:

signup.js, line 85

`changeNavigation()`

Updates navigation visuals to show the board section. This function removes the "clicked" class from summary and add task elements, and adds it to the board element, visually marking it as selected. Additionally, it adjusts element visibility based on screen size (potentially for mobile).

Source:

board_part1.js, line 53

`changeNavigationAddTask()`

Updates navigation visuals to show the "Add Task" section. This function switches the visual selection to the "Add Task" navigation item. Additionally, it adjusts element visibility based on screen size for mobile.

Source:

addTask_part1.js, line 33

`changeNavigationHighlightSummary()`

Highlights the summary navigation element and removes highlights from legal sections. This function adds the "navigation-item-clicked" class to the element with ID "navSummary", visually marking it as selected. It also removes the "navigation-legal-clicked" class from both legal notice and privacy policy elements. Additionally, it adjusts element visibility based on screen size (potentially for mobile).

Source:

summary.js, line 25

`changeNavigationPrivacyPolicy()`

Removes highlight from the navigation summary section. This function removes the "navigation-item-clicked" class from the element with ID "navSummary", likely deselecting it visually.

Source:

script.js, line 346

`changePriorityColor(buttonId)`

This function handles priority selection based on the clicked button's ID. It resets the `priorities` array, retrieves the priority value from the button's data attribute, and calls the appropriate styling function based on the button ID.

Parameters:

Name	Type	Description
`buttonId`	string	The ID of the clicked priority button (e.g., "urgentPriorityButton").

Source:

addTask_part1.js, line 110

checkAssignContact(i)

This function updates the visual representation of a contact in the "Assign To" dropdown menu to reflect its assigned state (selected). It targets specific elements based on dynamic IDs constructed using `dropdownEachContact(i)` and `assignToContactName(i)`. It sets the background color of the contact container, text color of the contact name, and calls `changeCheckBoxStyle` to update the checkbox image (likely to checked). Finally, it calls `showAvatarsOfSelectedContacts` to potentially update the assigned contacts avatar list.

Parameters:

Name	Type	Description
`i`	number	The index of the contact in the list.

Source:

addTask_part1.js, line 368

(async) checkCards()

Checks if there are any existing cards in the application. If there are no cards: - Restores default settings (implementation assumed in restoreDefault function) - Loads tasks again If there are cards, the function simply returns.

Source:

script.js, line 83

checkErrors(title, dueDate, category) → {boolean}

Checks for empty title, due date or invalid category.

Parameters:

Name	Type	Description
`title`	string	The task title.
`dueDate`	string	The task due date.
`category`	string	The task category.

Source:

board_part2.js, line 236

Returns:

True if all fields are valid, false otherwise.
Type

boolean

`checkForDuplicateEmail(email) → {boolean}`

This function checks if a contact with the provided email already exists in the `localContacts` array.

Parameters:

Name	Type	Description
`email`	string	The email address to check for duplication.

Source:

contacts_part2.js, line 8

Returns:

`true` if a duplicate email is found, `false` otherwise.
Type

boolean

`(async) checkIfEmailExists(email) → {Promise.<boolean>}`

Checks if an email already exists in the user data.

Parameters:

Name	Type	Description
`email`	string	Email address to check against the existing users.

Source:

login.js, line 6

Returns:

A promise that resolves to true if the email exists, otherwise false.
Type

Promise.<boolean>

`(async) checkIfUserIsAddedAsContact() → {Promise.<boolean>}`

Checks if the current user is already added as a contact in the local contacts list.

Source:

contacts_part1.js, line 47

Returns:

A promise that resolves with true if the user is a contact, or false otherwise.
Type

Promise.<boolean>

`checkInputValue()`

This function checks the value entered in the "Add Subtask" input field. It retrieves the trimmed value from the input field element. If the input value is empty, it calls `showDefaultInputMenu` (presumably to hide subtask options). Otherwise, it calls `showSubtaskInputMenu` (presumably to show options for subtasks).

Source:

addTask_part2.js, line 122

`checkPasswordStrength() → {boolean}`

Checks the strength of the password entered by the user.

Source:

signup.js, line 26

Returns:

Returns true if the password meets the strength criteria, otherwise rturns false.
Type

boolean

`checkPrivacyPolicy() → {boolean}`

Checks if the Privacy Policy checkbox is checked before final signup is possible If the checkbox is not checked, displays an alert message prompting the user to accept the Privacy Policy.

Source:

signup.js, line 108

Returns:

Returns true if the Privacy Policy checkbox is checked, otherwise returns false.
Type

boolean

`clearAddContactForm()`

This function clears the input fields in the "Add Contact" form.

Source:

contacts_part2.js, line 79

`clearPasswordError()`

Clears the error state and message when the password field is focused.

Source:

login.js, line 45

`clearSubtaskInputField()`

This function clears the value from the "Add Subtask" input field. It sets the value of the input field element to an empty string. It also calls `showDefaultInputMenu` (presumably to hide subtask options).

Source:

addTask_part2.js, line 136

clickedLegalPart()

Attaches click event listeners to all legal section navigation elements within the ".navigation-legal" container. When a legal section navigation element is clicked: - The previously clicked legal section element (if any) loses the "navigation-legal-clicked" class. - The clicked element gains the "navigation-legal-clicked" class for visual selection. - **Additionally:** The function calls `removeNavigationClick` to potentially remove click listeners from the main navigation items. (This behavior might need adjustment depending on your specific requirements.)

Source:

script.js, line 297

closeAssignToDropdown()

This function closes the "Assign To" dropdown menu. It sets the placeholder text back to "Select contacts to assign" and hides the dropdown container.

Source:

addTask_part1.js, line 235

closeCard()

Closes the Popup modal.

Source:

board_part2.js, line 345

closeContactInfo()

This function closes the contact info panel by removing the corresponding class from the DOM element.

Source:

contacts_part2.js, line 196

`closeDropdowenTask() → {void}`

Closes the task dropdown menu.

Source:

board_part1.js, line 278

Returns:

(nothing returned)
Type

void

`closeDropdown()`

Closes the navigation overlay dropdown menu. This function adds the 'd-none' class to the container element, likely hiding it.

Source:

script.js, line 190

`closeTaskCategoryDropdown()`

This function closes the "Select Task Category" dropdown menu. It hides the dropdown container.

Source:

addTask_part2.js, line 56

`createCardObject(id, place, category, categoryColor, title, description, dueDate, subtasks, assigned, priority, priorityImg) → {object}`

Creates a new card object with specified properties.

Parameters:

Name	Type	Description
`id`	number	The task ID.
`place`	string	The task placement.
`category`	string	The task category name.
`categoryColor`	string	The task category color.
`title`	string	The task title.
`description`	string	The task description.
`dueDate`	string	The task due date.
`subtasks`	array	An array of subtasks.
`assigned`	array	An array of assigned contacts.
`priority`	string	The task priority level (Urgent, Medium, Low).
`priorityImg`	string	The image path for the priority level.

Source:

board_part2.js, line 322

Returns:

The newly created card object.
Type

object

`(async) createCategories() → {Promise.<Object>}`

Creates categories for contacts based on the first letter of their names.

Source:

contacts_part1.js, line 127

Returns:

A promise that resolves with an object where keys are category initials (uppercase letters) and values are arrays of contacts belonging to that category.

Type

Promise.<Object>

(async) `createContact()` → {Promise.<boolean>}

This function creates a new contact, checks for duplicate emails, and performs subsequent actions such as updating storage, initializing the contact list, and opening the created contact's info.

Source:

contacts_part1.js, line 291

Returns:

A promise that resolves to `true` if a duplicate email is found, preventing further processing.
Type

Promise.<boolean>

`createNewContactDataSet(contactData, formattedName, existingContact)` → {Object}

This function creates a new contact data set object containing formatted name, initials, email, phone, category, and avatar color.

Parameters:

Name	Type	Default	Description
`contactData`	Object		An object containing name, email, and phone properties.
`formattedName`	Object		An object containing firstName and lastName properties.
`existingContact`	Object	null	(optional) An existing contact object to inherit avatar color from.

Source:

contacts_part1.js, line 396

Returns:

The newly created contact data set object.
Type

Object

`createTask()`

This function is the main entry point for creating a new task. It gathers data from the form, validates it, and builds a new task object. If validation fails, it displays error messages and exits. Otherwise, it adds the new task to the board, resets the form, and displays a success popup.

Source:

addTask_part2.js, line 262

`(async) createUserAsContact() → {Promise.<void>}`

Creates a new contact object for the current user if they are not already added.

Source:

contacts_part1.js, line 61

Returns:

A promise that resolves when the user is added as a contact.
Type

Promise.<void>

`(async) deleteContact(index) → {Promise.<void>}`

This function attempts to delete a contact at the specified index. Deletion is prevented if the contact's email matches the user's email.

Parameters:

Name	Type	Description
`index`	number	The index of the contact to delete in the `localContacts` array.

Source:

contacts_part2.js, line 358

Returns:

A promise that resolves when the deletion process is complete.
Type

Promise.<void>

`deleteCreatedSubtask(subTastIndex)`

This function removes a created subtask from the list. It removes the subtask at the provided index from the `createdSubtasks` array using the `splice` method. It then calls `openCreatedSubtaskBox` to refresh the displayed list of subtasks.

Parameters:

Name	Type	Description
`subTastIndex`	number	The index of the subtask to delete in the `createdSubtasks` array.

Source:

addTask_part2.js, line 221

`deleteTask(cardId)`

Removes a card from the cards array and the board based on its ID.

Parameters:

Name	Type	Description
`cardId`	number	The ID of the card to be deleted.

Source:

board_part2.js, line 364

`deleteUneditTask(id) → {void}`

Removes a task from the cards array by its ID.

Parameters:

Name	Type	Description
`id`	number	The ID of the task to remove.

Source:

board_part2.js, line 297

Returns:

(nothing returned)
Type

void

`displayClosestDueDate()`

Finds and displays the closest upcoming due date or overdue cards. This function: - Creates arrays for overdue and upcoming cards. - Calls `separateCards` to separate cards based on due dates and current date. - Calls `sortUpcomingCards` to sort upcoming cards by due date (if any). - Calls `updateDueDateContainers` to update the UI with the closest upcoming due date or overdue cards information.

Source:

summary.js, line 184

`doNotClose(event)`

Prevents the close event from bubbling up when clicking inside the big card modal.

Parameters:

Name	Type	Description
`event`	Event	The event object.

Source:

board_part2.js, line 32

`doneCardUpdate(done)`

Updates the cards in the specified section of the board ("todo", "progress", etc.).

Parameters:

Name	Type	Description
`done`	string	The name of the board section to update.

Source:

board_part1.js, line 156

`doneNumber()`

Calculates and displays the number of tasks in the "done" list. Performs error handling if the container element with ID 'done-number' is not found.

Source:

summary.js, line 120

`drop(place)`

Updates the card's "place" property in the cards array based on the drop target.

Parameters:

Name	Type	Description
`place`	string	The name of the drop target area (e.g., "todo", "progress").

Source:

board_part1.js, line 241

`dropMobile(place)`

Updates the card's "place" property in the cards array based on the drop target.

Parameters:

Name	Type	Description
`place`	string	The name of the drop target area (e.g., "todo", "progress").

Source:

board_part1.js, line 253

`dropdowenTask() → {void}`

Opens the task dropdown menu.

Source:

board_part1.js, line 267

Returns:

(nothing returned)
Type

void

`dropdownHelp()`

Loads the help template and closes the navigation overlay dropdown menu.

Source:

script.js, line 198

`(async) dropdownLegalNotice()`

Loads legal notice template and highlights the legal notice section. This function uses `async` to load the legal notice template via `Templates` (not provided). After successful loading (assumed), it: - Closes the dropdown (implementation in `closeDropdown` not provided). - Highlights the legal notice section with `legalNoticeHiglite`.

Source:

privacyAndLegal.js, line 10

`(async) dropdownPrivacyPolicy()`

Loads privacy policy template and highlights the privacy policy section. This function uses `async` to load the privacy policy template via `Templates` (not provided). After successful loading (assumed), it: - Closes the dropdown (implementation in `closeDropdown` not provided). - Highlights the privacy policy section with `privacyPolicyHighlight`. Loads the legal notice template and closes the navigation overlay dropdown menu.

Source:

privacyAndLegal.js, line 37

`editContactDeleteAndSaveButtonLayoutHTMLTemplate(index)`

This function populates the edit contact card with the details of the clicked contact. It sets the avatar background color and initials, fills the name, email, and phone input fields with the contact's data, and sets focus on the name field after a short delay.

Parameters:

Name	Type	Description
`index`	number	The index of the clicked contact in the `contacts` array.

Source:

templates.js, line 450

`editCreatedSubtask(i)`

This function edits a created subtask when its corresponding element is clicked. It adds a class "eachSubtaskFocused" to the clicked subtask element (likely for styling). It retrieves the current subtask text from the `createdSubtasks` array based on the provided index. It updates the inner HTML of the clicked subtask element with the `editCreatedSubtaskHTMLTemplate` (presumably to show an edit input field). It then focuses the edit input field and selects all its content.

Parameters:

Name	Type	Description
`i`	number	The index of the subtask in the `createdSubtasks` array.

Source:

addTask_part2.js, line 184

`(async) editTask(id)`

Opens edit task popup, loads edit template, handles OK button and sets edit mode for "finish" button. Calls functions to pre-fill edit form data based on provided task ID.

Parameters:

Name	Type	Description
`id`	string	The ID of the task to edit.

Source:

board_part2.js, line 97

`editTaskDone(id) → {void}`

Edits a task with the given ID.

Parameters:

Name	Type	Description
`id`	number	The ID of the task to edit.

Source:

board_part2.js, line 194

Returns:

(nothing returned)
Type

void

`emptyArrays() → {void}`

Empties the priority, assigned contacts, and created subtasks arrays.

Source:

board_part2.js, line 284

Returns:

(nothing returned)

Type

void

`errorMessageIfEmptyCategory()`

This function checks if a task category has been selected and displays an error message if not. It retrieves the text content of the "selectTaskCategoryTextField" element (presumably showing the selected category name). It selects the error message element using a query selector. If the text content is still "Select task category" (indicating no selection), it shows the error message and calls `highlightErrorMessage` for an animation effect. Otherwise, it hides the error message.

Source:

addTask_part2.js, line 82

`errorMessageIfEmptyDueDate() → {string|undefined}`

This function checks if the "Add Task" due date input field is empty and displays an error message if so. It also handles hiding the error message if the field is filled.

Source:

addTask_part1.js, line 89

Returns:

If the due date field is not empty, the function returns its value. Otherwise, it returns for error cases is needed).
Type

string | undefined

`errorMessageIfEmptyTitle() → {string|boolean}`

This function checks if the title input field is empty or contains only whitespace characters.

Source:

addTask_part1.js, line 67

Returns:

Returns the valid title if it's not empty or whitespace, otherwise returns false.
Type

string | boolean

`feedbackCardUpdate(feedback)`

Updates the cards in the specified section of the board ("todo", "progress", etc.).

Parameters:

Name	Type	Description
`feedback`	string	The name of the board section to update.

Source:

board_part1.js, line 133

`feedbackNumber()`

Calculates and displays the number of tasks in the "feedback" list. Performs error handling if the container element with ID 'feedback-number' is not found.

Source:

summary.js, line 99

`formatContactName(contactData) → {Object}`

This function formats a contact name by capitalizing the first letter of each word.

Parameters:

Name	Type	Description
`contactData`	Object	An object containing the name property to be formatted.

Source:

contacts_part1.js, line 361

Returns:

An object containing the formatted firstName and lastName properties.
Type

Object

`formatDueDate(dueDate) → {string}`

Formats the due date as a localized string.

Parameters:

Name	Type	Description
`dueDate`	string	The due date string in the format "YYYY-MM-DD".

Source:

summary.js, line 245

Returns:

- The formatted due date as a localized string.
Type

string

`getCategoryColor()`

This function takes the selected category name and finds the corresponding category object from the `taskCategories` array. If a match is found, it returns the category color. Otherwise, it logs an error message.

Source:

addTask_part2.js, line 365

`getContactData(name, email, phone) → {Object}`

This function extracts contact data (name, email, phone) from input fields or provided defaults.

Parameters:

Name	Type	Description
`name`	string	(optional) The name to use. If omitted, retrieves from the editContactName field.
`email`	string	(optional) The email address to use. If omitted, retrieves from the editContactEmail field.
`phone`	string	(optional) The phone number to use. If omitted, retrieves from the editContactPhone field.

Source:

contacts_part1.js, line 338

Returns:

An object containing the extracted name, email, and phone data.
Type

Object

`getContactRelatedInfo(contact) → {Object}`

This function retrieves contact-related information such as full name, initials, avatar color, and index in the selected assigned contacts array.

Parameters:

Name	Type	Description
`contact`	Object	The contact object.

Source:

addTask_part1.js, line 333

Returns:

An object containing full name, initials, avatar color, and index.
Type

Object

`getIndexByNameSurname(localContacts, firstName, lastName) → {number}`

This function finds the index of a contact in the `localContacts` array by name (first and last).

Parameters:

Name	Type	Description
`localContacts`	Array	The array of contact objects.
`firstName`	string	The first name of the contact to find.
`lastName`	string	The last name of the contact to find.

Source:

contacts_part2.js, line 29

Returns:

The index of the contact in the array, or -1 if not found.
Type

number

`getOldestOverdueDate(overdueCards) → {string}`

Finds the oldest overdue date from the provided array of overdue cards.

Parameters:

Name	Type	Description
`overdueCards`	Array	An array of overdue card objects.

Source:

summary.js, line 256

Returns:

- The oldest overdue date formatted as a localized string, or an empty string if no overdue cards exist.
Type

string

`getPriorityImagePath()`

This function takes the priority urgency level (e.g., 'Urgent', 'Medium', 'Low') and returns the corresponding image path for the priority icon.

Source:

addTask_part2.js, line 351

`getQueryParam(param) → {string|null}`

Retrieves the value of a specified URL parameter.

Parameters:

Name	Type	Description
`param`	string	The name of the URL parameter to retrieve.

Source:

privacyAndLegal.js, line 61

Returns:

The value of the URL parameter if found, otherwise null. This function uses the URLSearchParams interface to handle query string parameters. 'window.location.search' gives the query string part of the URL.
Type

string | null

`getSelectedContact(i, search) → {Object}`

This function retrieves the correct contact based on the search value.

Parameters:

Name	Type	Description
`i`	number	The index of the contact in the list.

`search`	string	The search value from the assignContactsDropdown input.

Source:

addTask_part1.js, line 320

Returns:

The selected contact object.
Type

Object

`getTaskData()`

This function gathers data from the create task form and returns an object containing the task details. It retrieves title, due date, category, assigned contacts, description, and subtasks using relevant functions.

Source:

addTask_part2.js, line 320

`giveId() → {number}`

Generates a unique ID for a new card.

Source:

addTask_part2.js, line 282

Returns:

A unique ID for the new card.
Type

number

`goBack()`

Redirects the user back to their original entry page. This function determines whether the user originally came from the 'login' or 'signup' page by checking the 'ref' URL parameter and redirects them back to that page. It provides a convenient way for users to return to their previous context after visiting a linked page, like a Privacy Policy or Legal Notice.

Source:

privacyAndLegal.js, line 86

`greetUser()`

Displays a greeting message based on the current time of day to the logged-in user.

Source:

login.js, line 205

`greetUserMobile()`

Displays a greeting message based on the current time of day to the logged-in user.

Source:

script.js, line 113

`(async) guestLogin()`

/** Sets up the session for a guest user and calls greetUser to display a welcome message.

Source:

login.js, line 183

`handleContactAssignment(contactIndex, fullName, initials, avatarColor, i)`

This function handles the assignment or unassignment of a contact based on the contact index.

Parameters:

Name	Type	Description

`contactIndex`	number	The index of the contact in the selectedAssignedContacts array.
`fullName`	string	The full name of the contact.
`initials`	string	The initials of the contact.
`avatarColor`	string	The avatar color of the contact.
`i`	number	The index of the contact in the list.

Source:

addTask_part1.js, line 349

`handleRememberMeChange()`

Handles the change event for the "Remember Me" checkbox. Fetches email from the DOM and toggles the remember password setting based on the checkbox state.

Source:

login.js, line 98

`hideAddContactCard()`

Hides the "Add Contact" card by: 1. Animating the card exit with a slight delay. 2. Hiding the container element after the animation.

Source:

contacts_part1.js, line 276

`hideContactEditDeleteMenu()`

This function hides the edit and delete menu container by removing the corresponding class.

Source:

contacts_part2.js, line 376

`highlightCreatedContact(index)`

This function visually highlights the newly created contact in the contact list.

Parameters:

Name	Type	Description
`index`	number	The index of the newly created contact.

Source:

contacts_part2.js, line 60

`highlightErrorMessage(errorMessage)`

This function animates the error message element to highlight it briefly. It sets an animation style for 1 second and then removes the animation style after a short delay.

Parameters:

Name	Type	Description
`errorMessage`	HTMLElement	The error message element to animate.

Source:

addTask_part2.js, line 99

`(async) includeAddTask()`

Fetches and includes the content of external HTML templates marked with the 'include-AddTask' attribute.

Source:

board_part2.js, line 61

`(async) includeHTML()`

Fetches and includes the content of external HTML templates marked with the 'include-html' attribute.

Source:

script.js, line 143

(async) init()

Initializes the application after a startup animation. This function checks if a user is logged in (based on localStorage). If not, it inserts an animation and redirects to the login page after 1.2 seconds. Then, it performs various asynchronous tasks in sequence: - Loads user data - Displays a mobile greeting (if applicable) - Loads tasks - Checks for existing cards (potentially for restoring defaults) - Loads summary information - Greets the user (potentially with a different greeting for mobile)

Source:

script.js, line 14

(async) initContacts() → {Promise.<void>}

Initializes the contact display by performing the following steps: 1. Sorts contacts by first name. 2. Creates categories for contacts based on their first letter. 3. Renders the contact list with categories and individual contacts.

Source:

contacts_part1.js, line 106

Returns:

A promise that resolves when the contact list is initialized.
Type

Promise.<void>

insertAnimation()

Inserts an animation overlay element into the DOM. This function creates an HTML string representing an overlay element with a logo image. It then inserts this HTML content at the beginning of the document body using `insertAdjacentHTML`.

Source:

script.js, line 47

`isAssignedEdit(card)`

Pre-fills assigned contact information based on the provided card's assigned contacts.

Parameters:

Name	Type	Description
`card`	object	The task object containing assigned contact information.

Source:

board_part2.js, line 147

`legalNoticeHiglite()`

Highlights the legal notice navigation element. This function adds the "navigation-legal-clicked" class to the element with ID "navLegalNotice", visually marking it as selected.

Source:

privacyAndLegal.js, line 23

`(async) loadAddTasks()`

Asynchronously loads "add task" templates and initializes functionalities. This function uses `async` to load templates via `Templates`. After successful loading, it: - Initializes "add task" functionalities with `addTaskInit`. - Updates navigation to show "Add Task" with `changeNavigationAddTask`.

Source:

addTask_part1.js, line 9

`(async) loadBoard()`

Asynchronously loads all necessary functions for the board in the correct order.

Source:

board_part1.js, line 40

`(async) loadContacts() → {Promise.<void>}`

Loads contacts by rendering templates, fetching remote contacts, checking if the user is added, and initializing the display.

Source:

contacts_part1.js, line 14

Returns:

A promise that resolves when all contacts are loaded and displayed.
Type

Promise.<void>

`(async) loadRememberedPassword()`

Automatically fills in the password field and checks the "Remember Me" checkbox if the user's email is found and remembered.

Source:

login.js, line 137

`(async) loadRemoteContactsOfLoggedInUser() → {Promise.<Array.<Object>>}`

Fetches remote contacts for the logged-in user from storage.

Source:

contacts_part1.js, line 24

Returns:

A promise that resolves with an array of contact objects, or an empty array if no contacts are found.
Type

Promise.<Array.<Object>>

`(async) loadTasks() → {Promise.<Array.<object>>}`

Loads tasks from local storage.

Source:

board_part1.js, line 11

Returns:

A promise that resolves to an array of tasks or an empty array if no tasks are found.
Type

Promise.<Array.<object>>

`(async) loadUsers() → {Promise.<Array>}`

Asynchronously loads user data from storage.

Source:

signup.js, line 122

Throws:

Throws an error if there is an issue loading the users.

Type

Error

Returns:

A promise that resolves to an array of users if found, otherwise returns an empty array.
Type

Promise.<Array>

(async) `login()` → {Promise.<void>}

Attempts to log in the user by comparing the provided credentials with stored users. Sets local storage items if the credentials are valid, otherwise displays an error message.

Source:

login.js, line 17

Returns:

Type

Promise.<void>

`logout()`

Logs out the current user by clearing session-related data and redirecting to the login page.

Source:

login.js, line 248

`lowPriorityButtonStylingWhenClicked(button)`

This function applies styling for the clicked low priority button, highlighting it and resetting styles for other priority buttons.

Parameters:

Name	Type	Description
`button`	Element	The clicked low priority button element.

Source:

addTask_part1.js, line 156

`matchingCategoryCheck(matchingCategory)` → {string}

Retrieves category color based on matching category object.

Parameters:

Name	Type	Description
`matchingCategory`	object | null	The matching category object (if found).

Source:

board_part2.js, line 270

Returns:

The category color (if match found), otherwise logs an error.
Type

string

`mediumPriorityButtonStylingWhenClicked(button)`

This function applies styling for the clicked medium priority button, highlighting it and resetting styles for other priority buttons.

Parameters:

Name	Type	Description
`button`	Element	The clicked medium priority button element.

Source:

addTask_part1.js, line 143

`(async) mobileGreeting()`

Displays a greeting and potentially performs additional actions for mobile users. This function checks the window inner width to determine if the user is on a mobile device. If the width is less than 800 pixels: - It calls the assumed `TemplateGreetMobile` function (likely for mobile greeting display) - It calls `greetUserMobile` (assumed to personalize the greeting for mobile) - It waits for 1.2 seconds using a Promise with setTimeout

Source:

script.js, line 102

`navigateTo(page)`

Redirects the user to a specified page while retaining the 'ref' URL parameter.

Parameters:

Name	Type	Description
`page`	string	The relative URL to which the user should be redirected. This function is primarily used to navigate between related pages (like Privacy Policy and Legal Notice) while keeping track of the user's original entry page (e.g., 'login' or 'signup'). It appends the 'ref' parameter to the URL to maintain the reference throughout the navigation.

Source:

privacyAndLegal.js, line 74

`navigationClick()`

Attaches click event listeners to all navigation items (.navigation-item class). When a navigation item is clicked: - The previously clicked item (if any) loses the "navigation-item-clicked" class. - The clicked item gains the "navigation-item-clicked" class for visual selection. - All navigation items are reset (hidden clicked images, shown unclicked images). - The `navigationClickImg` function is called to potentially change the clicked item's image (if screen is below 800px).

Source:

script.js, line 235

`navigationClickImg()`

Handles potential image change for the currently clicked navigation item (if screen is below 800px). This function retrieves the element with the class "navigation-item-clicked" (the clicked item). If the clicked item exists and the screen width is less than 800px: - It finds the "unclicked" and "clicked.d-none" images within the clicked item. - It hides the "unclicked" image using `classList.add('d-none')`. - It shows the "clicked" image using `classList.remove('d-none')`.

Source:

script.js, line 273

`newContactDataSetForArray(name, email, phone) → {Object}`

This function creates a new contact data set object by: 1. Extracting contact data (name, email, phone) from input fields or provided defaults. 2. Formatting the contact name by capitalizing the first letter of each word. 3. Creating a new contact data set with formatted name, initials, email, phone, category, and avatar color.

Parameters:

Name	Type	Description
`name`	string	(optional) The name to use. If omitted, retrieves from the editContactName field.
`email`	string	(optional) The email address to use. If omitted, retrieves from the editContactEmail field.
`phone`	string	(optional) The phone number to use. If omitted, retrieves from the editContactPhone field.

Source:

contacts_part1.js, line 319

Returns:

An object containing the contact data set with formatted name, initials, email, phone, category, and avatar color.

Type

Object

`openAssignToDropdown()`

This function opens the "Assign To" dropdown menu. It clears the placeholder text, sets the display to flex, clears the inner HTML, scrolls down the container to ensure visibility, and renders all contacts.

Source:

addTask_part1.js, line 194

`openContactInfo(index)`

This function opens the contact info card and populates it with the contact at the specified index.

Parameters:

Name	Type	Description
`index`	number	The index of the contact to open.

Source:

contacts_part2.js, line 43

`openContactInfoHTMLTemplate(index)`

This function populates the contact info panel with the details of the clicked contact. It retrieves the contact data from the `contacts` array using the provided index. Then, it updates the HTML content of specific DOM elements with the contact's name, initials, avatar color, email, and phone number. Finally, it updates the "Edit Contact" and "Delete Contact" buttons with the clicked contact's index for proper functionality.

Parameters:

Name	Type	Description
`index`	number	The index of the clicked contact in the `contacts` array.

Source:

templates.js, line 329

`openCreatedSubtaskBox()`

This function opens the box that displays the created subtasks. It shows the container element for the created subtasks and clears its inner HTML. It then loops through each created subtask in the `createdSubtasks` array. For each subtask, it calls `openCreatedSubtaskBoxHTMLTemplate` (presumably to generate the HTML for the subtask) and adds it to the container's inner HTML. Finally, it calls `clearSubtaskInputField` to clear the input field.

Source:

addTask_part2.js, line 165

`openDropdown()`

Opens the navigation overlay dropdown menu. This function removes the 'd-none' class from the container element, likely making it visible.

Source:

script.js, line 180

`openTaskCategoryDropdown()`

This function opens the "Select Task Category" dropdown menu. It shows the dropdown container, clears its inner HTML, and loops through each task category. For each category, it constructs the dropdown entry HTML with the category name and sets an onclick event listener to call `selectTaskCategory` when clicked. Finally, it calls `scrollDown` (presumably to ensure visibility).

Source:

addTask_part2.js, line 38

`outputDeadlineText(deadlineText)`

Updates the "deadline" element text content (if it exists). This function retrieves the DOM element with ID "deadline" and updates its text content with the provided `deadlineText` string.

Parameters:

Name	Type	Description
`deadlineText`	string	The text to display in the "deadline" element.

Source:

summary.js, line 338

`outputDueDate(output)`

Updates the "due-date" element text content (if it exists). This function retrieves the DOM element with ID "due-date" and updates its text content with the provided `output` string.

Parameters:

Name	Type	Description
`output`	string	The text to display in the "due-date" element.

Source:

summary.js, line 324

`priorityEdit(card)`

Sets the priority button color based on the provided card's urgency level.

Parameters:

Name	Type	Description
`card`	object	The task object containing urgency information.

Source:

board_part2.js, line 132

`priorityImgCheck(priority) → {string}`

Maps priority level to corresponding image path.

Parameters:

Name	Type	Description
`priority`	string	The task priority (Urgent, Medium, Low).

Source:

board_part2.js, line 253

Returns:

The image path for the priority level.
Type

string

`privacyPolicyHighlight()`

Highlights the privacy policy navigation element. This function adds the "navigation-legal-clicked" class to the element with ID "navPrivacyPolicy", visually marking it as selected. Loads the privacy policy template and closes the navigation overlay dropdown menu.

Source:

privacyAndLegal.js, line 49

`progressCardUpdate(progress)`

Updates the cards in the specified section of the board ("todo", "progress", etc.).

Parameters:

Name	Type	Description
`progress`	string	The name of the board section to update.

Source:

board_part1.js, line 110

`progressNumber()`

Calculates and displays the number of tasks in the "progress" list. Performs error handling if the container element with ID 'progress-task-number' is not found.

Source:

summary.js, line 78

`progressbarCompetedRate(card) → {number}`

Calculates the completion percentage for a card's progress bar based on the number of completed subtasks.

Parameters:

Name	Type	Description
`card`	object	The card object containing the subtasks list.

Source:

board_part1.js, line 206

Returns:

The percentage of completed subtasks (0-100).
Type

number

`redesignAddContactCardToEditContactCard()`

This function visually changes the add contact card to the edit contact card layout. It adds the "show" class for animation, changes the title to "Edit contact", and hides the subtitle.

Source:

contacts_part2.js, line 325

`(async) rememberPassword(email, password, remember) → {Promise.<void>}`

Updates the user's password in local storage if the "Remember Me" checkbox is checked.

Parameters:

Name	Type	Description
`email`	string	User's email address to identify the user.
`password`	string	Password to be remembered.
`remember`	boolean	Flag to determine whether to remember or forget the password.

Source:

login.js, line 112

Returns:

Type

Promise.<void>

`removeClickedLegalPart()`

Attaches click event listeners to all navigation items (.navigation-item class). When a navigation item is clicked, this function removes the "navigation-legal-clicked" class from any legal section element that might have it. - This ensures clicking a main navigation item clears the "clicked" state for legal section elements.

Source:

script.js, line 331

`removeNavHighlightLegalPartOnDropdown()`

Removes highlight from any clicked legal section element on dropdown click. This function is similar to `removeNavHighlightOnDropdown` but targets elements with the "navigation-legal-clicked" class. When a dropdown anchor is clicked, it removes the "navigation-legal-clicked" class from any currently highlighted legal section element (if any).

Source:

script.js, line 375

`removeNavHighlightOnDropdown()`

Removes highlight from any clicked navigation item on dropdown click. This function attaches click event listeners to all anchor tags within elements with the class "dropdown-container". When a dropdown anchor is clicked, it removes the "navigation-item-clicked" class from any currently highlighted navigation item (if any).

Source:

script.js, line 358

`removeNavigationClick()`

Attempts to remove click event listeners from all navigation items (.navigation-item class). This function loops through all elements with the class ".navigation-item". It attempts to remove any existing click event listeners from these elements. - Note that this function might not always successfully remove listeners depending on how they were previously attached.

Source:

script.js, line 315

`renderAddContactLayout()`

Renders the initial layout for adding a contact within the card. - Sets the headline to "Add contact". - Shows the subheadline element. - Sets the avatar icon background color and adds an "Add Contact" image. - Clears the input fields for name, email, and phone. - Updates the buttons section with the HTML template for cancel and create buttons.

Source:

contacts_part1.js, line 242

`renderAllContacts()`

This function renders all contacts from the `contacts` array in the "Assign To" dropdown menu. It first sorts the contacts by first name and then loops through each contact. For each contact, it checks if it's already assigned (based on `selectedAssignedContacts`). It then sets the background color, text color, and checkbox source based on the assigned status. Finally, it adds the contact HTML template to the dropdown container.

Source:

addTask_part1.js, line 218

`renderContactCategory(initial) → {string}`

Renders the HTML structure for a single contact category (e.g., "A").

Parameters:

Name	Type	Description
`initial`	string	The first letter (uppercase) representing the category.

Source:

contacts_part1.js, line 181

Returns:

The HTML string for the contact category.
Type

string

`renderContactCategoryAndEachContact(categories, contactListHTML, index) → {string}`

Recursively renders the HTML for contact categories and individual contacts within those categories.

Parameters:

Name	Type	Description

`categories`	Object	An object containing contact categories (initials as keys, contact arrays as values).
`contactListHTML`	string	The accumulated HTML string for the contact list.
`index`	number	A counter to keep track of unique IDs for each contact.

Source:

contacts_part1.js, line 164

Returns:

The updated `contactListHTML` string with rendered categories and contacts.
Type

string

`(async) renderContactList(categories) → {Promise.<void>}`

Renders the HTML structure for the contact list with categories and individual contacts.

Parameters:

Name	Type	Description
`categories`	Object	An object containing contact categories (initials as keys, contact arrays as values).

Source:

contacts_part1.js, line 145

Returns:

A promise that resolves when the contact list is rendered.
Type

Promise.<void>

`renderEachContact(contact, index) → {string}`

Renders the HTML structure for a single contact with its details.

Parameters:

Name	Type	Description
`contact`	Object	A contact object containing properties like name, surname, email, avatarColor, and initials.
`index`	number	A unique identifier for the contact.

Source:

contacts_part1.js, line 198

Returns:

The HTML string for the individual contact.
Type

string

`renderFilteredContacts(filteredContacts)`

This function renders a list of filtered contacts in the "Assign To" dropdown menu. It takes an array of filtered contacts as input. It shows the dropdown container, clears its inner HTML, and loops through each filtered contact. Similar to `renderAllContacts`, it checks the assignment status, sets styles, and adds the contact HTML template (presumably generated by another function `renderFilteredContactsHTMLTemplate`) to the dropdown container.

Parameters:

Name	Type	Description
`filteredContacts`	array	An array of contacts matching the search criteria.

Source:

addTask_part1.js, line 289

`resetAddTaskForm()`

This function resets the form fields for creating a new task. It clears the values of the following input elements: - addTaskInputTitle (presumably for the task title) - addTaskDescriptionInput (presumably for the task description) - addTaskDueDateInput (presumably for the task due date) It calls `changePriorityColor` to reset the priority color selection (presumably to a default value like 'mediumPriorityButton'). It clears the inner HTML of the element with the ID 'avatarsOfSelectedContacts' (likely to remove any displayed assigned contacts). It resets the `selectedAssignedContacts` array to an empty array. It updates the text content of the element with the ID 'selectTaskCategoryTextField' to "Select task category" (presumably to reset the selected category). It resets the `createdSubtasks` array to an empty array (presumably to remove any created subtasks). Finally, it hides any error messages related to empty title, due date, or category selection using query selectors.

Source:

addTask_part2.js, line 239

`resetContactButtonColor(buttonElement, buttonId)`

This function resets the color and style of a clicked contact button to its original state. It also removes the button's ID from the `clickedButtons` array.

Parameters:

Name	Type	Description
`buttonElement`	HTMLElement	The DOM element of the button to reset.
`buttonId`	string	The ID of the button to reset.

Source:

contacts_part2.js, line 168

`resetContactButtonEmailColor(emailElement, emailId)`

This function resets the color of a clicked contact's email element to its original state. It also removes the email element's ID from the `clickedButtonEmailColors` array.

Parameters:

Name	Type	Description
`emailElement`	HTMLElement	The DOM element of the email element to reset.
`emailId`	string	The ID of the email element to reset.

Source:

contacts_part2.js, line 243

`resetCreateTaskFormInputs()`

This function resets the input fields and state of the create task form. It clears the `boardPlace` variable (presumably holding the selected board location), resets the `priorities` array (likely containing available priorities), clears the `selectedAssignedContacts` array (presumably holding selected contacts), and empties the `createdSubtasks` array (likely containing created subtasks).

Source:

addTask_part2.js, line 417

`resetLastClickedContactButtonColor()`

This function resets the color of the previously clicked contact button (if any). It retrieves the ID of the last clicked button from the `clickedButtons` array and removes it from the DOM styles.

Source:

contacts_part2.js, line 185

`resetLastClickedContactButtonEmailColor()`

This function resets the color of the previously clicked contact's email element (if any). It retrieves the ID of the last clicked email element from the `clickedButtonEmailColors` array and removes it from the DOM styles.

Source:

contacts_part2.js, line 260

`resetNavigationItems()`

Resets all navigation items to their initial state. This function loops through all elements with the class ".navigation-item". Within each item, it: - Hides any previously shown "clicked" image (adds the "d-none" class). - Shows any previously hidden "unclicked" image (removes the "d-none" class).

Source:

script.js, line 257

`resetOtherPriorityButtonStyles(buttonId, iconId)`

This function resets styling for a specified priority button and its icon.

Parameters:

Name	Type	Description
`buttonId`	string	The ID of the button to reset styles for.
`iconId`	string	The ID of the icon element associated with the button.

Source:

addTask_part1.js, line 169

`saveEditSubtaskInput(i)`

This function saves the edited subtask when the user confirms changes in the edit input field. It removes the "eachSubtaskFocused" class from the edited subtask element (likely for styling). It retrieves the trimmed value from the edit input field element. If the edited value is not empty, it updates the inner HTML of the edited subtask element with the `saveEditSubtaskInputHTMLTemplate` (presumably to show the updated text). It then updates the corresponding subtask object in the `createdSubtasks` array with the edited text and sets its "done" status to false (assuming it wasn't changed). Otherwise, if the edited value is empty, it calls `deleteCreatedSubtask` to remove the subtask.

Parameters:

Name	Type	Description
`i`	number	The index of the subtask in the `createdSubtasks` array.

Source:

addTask_part2.js, line 203

`(async) saveNewUser(user) → {Promise.<void>}`

Saves a new user to the storage.

Parameters:

Name	Type	Description
`user`	Object	The user object to be saved.

Source:

signup.js, line 220

Returns:

Type

Promise.<void>

`saveSubtaskInput()`

This function saves the entered subtask from the "Add Subtask" input field. It retrieves the trimmed value from the input field element. If the input value is not empty, it creates a new subtask object with the text and sets its "done" status to false. It then pushes the new subtask object to the `createdSubtasks` array. Finally, it calls `openCreatedSubtaskBox` to display the created subtasks and `scrollDown` (presumably to ensure visibility).

Source:

addTask_part2.js, line 148

`scrollDown()`

This function scrolls the element with the ID "addTaskContainer" down by 120 pixels.

Source:

addTask_part1.js, line 206

`scrollToAnchor(anchorId)`

This function smooth-scrolls the webpage to the element with the specified anchor ID.

Parameters:

Name	Type	Description
`anchorId`	string	The ID of the anchor element to scroll to.

Source:

contacts_part2.js, line 90

`search()`

Searches cards based on user input, hiding unmatched & showing matches. Calls NoMatchFound for empty search or no results.

Source:

board_part2.js, line 5

`searchContactToAssign()`

This function handles searching for contacts within the "Assign To" dropdown menu. It retrieves the search term from the "assignContactsDropdown" element and converts it to lowercase. If the search term is empty, it renders all contacts again. Otherwise, it filters the `contacts` array based on whether the name or surname (lowercase) starts with the search term. Finally, it renders the filtered contacts using the `renderFilteredContacts` function.

Source:

addTask_part1.js, line 268

`selectTaskCategory(i)`

This function handles selecting a task category from the dropdown menu. It retrieves the selected task category object based on the provided index. It updates the text content of the "selectTaskCategoryTextField" element with the selected category name. It then closes the dropdown and calls `errorMessageIfEmptyCategory` to check for an empty selection.

Parameters:

Name	Type	Description
`i`	number	The index of the selected task category in the `taskCategories` array.

Source:

addTask_part2.js, line 68

`separateCards(overdueCards, upcomingCards, currentDate)`

Separates cards into overdue and upcoming categories based on due dates. This function iterates through the `cards` array (assumed to be an array of card objects). For each card with a place other than "done": - It parses the `dueDate` string into a Date object (if it exists). - If the due date exists and is earlier than `currentDate`, it adds the card to `overdueCards`. - If the due date exists and is later than or equal to `currentDate`, it adds the card to `upcomingCards`.

Parameters:

Name	Type	Description
overdueCards	array	An array to store overdue cards.
upcomingCards	array	An array to store upcoming cards.
currentDate	Date	The current date object.

Source:

summary.js, line 208

setButtonColorAsClicked(buttonElement)

This function sets the color and style of a contact button to indicate it's clicked.

Parameters:

Name	Type	Description
buttonElement	HTMLElement	The DOM element of the button to style.

Source:

contacts_part2.js, line 177

setEmailColorAsClicked(emailElement)

This function sets the color of a contact's email element to white.

Parameters:

Name	Type	Description
emailElement	HTMLElement	The DOM element of the email element to style.

Source:

contacts_part2.js, line 252

setGuestLogin()

Sets up the session for a guest user by storing necessary data in local storage.

Source:

login.js, line 196

`setUserLogin(user)`

Sets up the user session in localStorage with the user's information.

Parameters:

Name	Type	Description
`user`	Object	The user object with at least a 'name' property.

Source:

login.js, line 74

`showAddContactCard()`

Shows the "Add Contact" card by: 1. Unhiding the container element. 2. An animating the card entrance with a slight delay. 3. Rendering the initial layout for adding a contact. 4. Preventing event bubbling on the card itself.

Source:

contacts_part1.js, line 223

`showAvatarsOfSelectedContacts()`

This function updates the list of assigned contacts' avatars displayed below the dropdown menu. It first sorts the `selectedAssignedContacts` array by name in ascending order using `localeCompare`. It then shows the container for the avatar list, clears its inner HTML, and loops through each assigned contact. For each contact, it constructs the avatar HTML element with the contact's initials and background color set to the contact's avatar color. Finally, it adds the avatar HTML to the container.

Source:

addTask_part1.js, line 398

`showContactCreatedPopUp()`

This function displays a temporary "Contact Created" pop-up notification.

Source:

contacts_part2.js, line 69

`showContactEditDeleteMenu(index)`

This function shows the edit and delete menu for the clicked contact on mobile screens (less than 800px wide). It populates the menu container with the "Edit" and "Delete" buttons and adds an event listener to prevent event bubbling.

Parameters:

Name	Type	Description
`index`	number	The index of the clicked contact in the `contacts` array.

Source:

templates.js, line 425

`showCurrentContactDetails(index)`

This function populates the edit contact card with the details of the clicked contact. It sets the avatar background color and initials, fills the name, email, and phone input fields with the contact's data, and sets focus on the name field after a short delay.

Parameters:

Name	Type	Description
`index`	number	The index of the clicked contact in the `contacts` array.

Source:

contacts_part2.js, line 337

`showEditAndDeleteMenuOnMobile()`

This function hides the edit and delete menu on mobile screens (less than 800px wide) when the edit contact card is opened.

Source:

contacts_part2.js, line 314

`showEditContact(index)`

This function opens the edit contact card to modify the details of the clicked contact. It shows the card container, sets a timeout to add the "show" class with animation, and calls other functions to handle additional functionalities.

Parameters:

Name	Type	Description
`index`	number	The index of the clicked contact in the `contacts` array.

Source:

contacts_part2.js, line 294

`showTaskCreatedPopUp()`

This function displays a popup notification for successfully creating a new task. It manipulates the styles of DOM elements with specific IDs to achieve the visual effect. - Sets the display of the container element (`taskCreatedButtonContainer`) to "flex". - Uses `setTimeout` to schedule adding the 'showTaskCreatedButtonContainer' class to the button element (`taskCreatedButton`) with a 20ms delay. - Uses another `setTimeout` to schedule removing the class and hiding the container element after 800ms. - Finally, it calls the `loadBoard` function (assumed to be defined elsewhere) with another 20ms delay, potentially to refresh the board view.

Source:

addTask_part2.js, line 432

`showTaskCreatedPopUpBoard() → {void}`

Displays a temporary popup board indicating task creation and reloads the board.

Source:

addTask_part2.js, line 451

Returns:

(nothing returned)
Type

void

(async) sortByFirstName() → {Promise.<void>}

Sorts the `localContacts` array in ascending order by the first name of each contact.

Source:

contacts_part1.js, line 117

Returns:

A promise that resolves when sorting is complete.
Type

Promise.<void>

sortUpcomingCards(upcomingCards) → {Array}

Sorts the upcoming cards array by due date in ascending order.

Parameters:

Name	Type	Description
`upcomingCards`	Array	An array of upcoming card objects.

Source:

summary.js, line 232

Returns:

- The sorted array of upcoming cards.
Type

Array

(async) startAnimation()

Simulates a startup animation with a delay before redirecting to the login page. This function uses a Promise to create an asynchronous delay. It sets a timeout of 1.2 seconds and then redirects the user to the login.html page. The Promise resolves after the timeout, allowing subsequent asynchronous operations in `init` to proceed.

Source:

script.js, line 63

`startDragging(id)`

Sets the `currentDraggedElement` variable to the ID of the card being dragged.

Parameters:

Name	Type	Description
`id`	number	The ID of the dragged card element.

Source:

board_part1.js, line 224

`subtaskCompleted(index, cardId)`

Updates the image of a subtask to show completion and marks it as completed internally.

Parameters:

Name	Type	Description
`index`	number	The index of the subtask within the card's subtasks array.
`cardId`	number	The ID of the card containing the subtask.

Source:

board_part1.js, line 422

`subtaskEdit(card)`

Handles pre-filling subtask data based on the provided card's subtasks. (Specific details depend on your subtask implementation)

Parameters:

Name	Type	Description

`card`	object	The task object containing subtask information.

Source:

board_part2.js, line 178

`subtaskNotCompleted(index, cardId)`

Updates the image of a subtask to show incompletion and marks it as incomplete internally.

Parameters:

Name	Type	Description
`index`	number	The index of the subtask within the card's subtasks array.
`cardId`	number	The ID of the card containing the subtask.

Source:

board_part1.js, line 405

`successfulLogin()`

Displays the login modal.

Source:

login.js, line 55

`successfulSignup()`

Displays the login modal.

Source:

signup.js, line 185

`(async) summaryLoad()`

Loads summary template, updates summary numbers, and highlights summary navigation. Loads and displays the summary template and then retrieves task counts for each list. This function uses `async` to load the summary template via `Templates` (not provided). After successful loading (assumed), it: - Updates summary numbers with `summaryLoadNumbers` (implementation not provided). - Highlights the summary navigation section with `changeNavigationHighlightSummary`.

Source:

summary.js, line 11

`summaryLoadNumbers()`

Calls functions to retrieve and display the number of tasks in each list ("todo", "progress", "feedback", "done").

Source:

summary.js, line 43

`templateOkBtn(id)`

Updates the content of the "createTaskContainerPopup" element with an "Ok" button for completing task editing.

Parameters:

Name	Type	Description
`id`	string	The ID of the task being edited (likely used for internal logic).

Source:

templates.js, line 298

`todoCardUpdate(todo)`

Updates the cards in the specified section of the board ("todo", "progress", etc.).

Parameters:

Name	Type	Description
`todo`	string	The name of the board section to update.

Source:

board_part1.js, line 87

`todoNumber()`

Calculates and displays the number of tasks in the "todo" list. Performs error handling if the container element with ID 'to-do-number' is not found.

Source:

summary.js, line 57

`toggleAssignToDropdown()`

This function toggles the visibility of the "Assign To" dropdown menu. If the dropdown is currently hidden, it will be opened. Otherwise, it will be closed.

Source:

addTask_part1.js, line 181

`toggleCheckbox(buttonElement)`

Toggles the checkbox that states if the Privacy Policy was accepted or not and updatates the checkbox image. Toggles the state of a checkbox and updates the image icon to checked or not checked This function is used on login and signup pages to handle user interaction with the checkboxes, such as remembering passwords and accepting privacy policies.

Parameters:

Name	Type	Description
`buttonElement`	HTMLElement	On click of this button, the state of the checkbox will be toggled.

Source:

login.js, line 87

`toggleContactButtonColor(buttonId, buttonElement, index)`

This function toggles the color and style of a clicked contact button. It checks if the button has already been clicked. - If clicked: - Resets the button's color and style to its original state. - Closes the contact info. - If not clicked: - Sets the button's color and style to indicate it's clicked. - Opens the contact info. - Adds the button's ID to the `clickedButtons` array. - Resets the color of the previously clicked contact button (if any).

Parameters:

Name	Type	Description
`buttonId`	string	The ID of the clicked button.
`buttonElement`	HTMLElement	The DOM element of the clicked button.
`index`	number	The index of the clicked contact in the `contacts` array.

Source:

contacts_part2.js, line 150

`toggleContactButtonEmailColor(emailId, emailElement, index)`

This function toggles the color of the email element within a clicked contact button. It checks if the email element has already been clicked. - If clicked: - Resets the email element's color to its original state. - If not clicked: - Sets the email element's color to white. - Adds the email element's ID to the `clickedButtonEmailColors` array. - Resets the color of the previously clicked contact's email element (if any).

Parameters:

Name	Type	Description
`emailId`	string	The ID of the email element within the clicked contact button.
`emailElement`	HTMLElement	The DOM element of the email element.
`index`	number	The index of the clicked contact in the `contacts` array.

Source:

contacts_part2.js, line 227

`togglePassword(fieldId)`

Toggles the visibility of the password input field and changes the visibility icon accordingly.

Parameters:

Name	Type	Description
`fieldId`	string	The ID of the password input field.

Source:

signup.js, line 67

`togglePrivacyPolicyCheckbox() → {boolean}`

Checks if the Privacy Policy checkbox is checked before final signup is possible. If the checkbox is not checked, displays an alert message prompting the user to accept the Privacy Policy.

Source:

signup.js, line 94

Returns:

Returns true if the Privacy Policy checkbox is checked, otherwise false.
Type

boolean

`toggleRememberMeCheckbox(label)`

Toggles the visibility and state of a custom checkbox UI element.

Parameters:

Name	Type	Description
`label`	HTMLElement	The label element associated with the checkbox.

Source:

login.js, line 128

`toggleSelectTaskCategoryDropdown()`

This function toggles the visibility of the "Select Task Category" dropdown menu. If the dropdown is currently hidden, it will be opened and `openTaskCategoryDropdown` is called. Otherwise, it will be closed and `errorMessageIfEmptyCategory` is called to check for an empty selection.

Source:

addTask_part2.js, line 22

`uncheckAssignContact(i)`

This function updates the visual representation of a contact in the "Assign To" dropdown menu to reflect its unassigned state (unselected). It targets specific elements based on dynamic IDs constructed using `dropdownEachContact(i)` and `assignToContactName(i)`. It sets the background color of the contact container and text color of the contact name back to defaults. It calls `changeBackCheckBoxStyle` to update the checkbox image (likely to unchecked). Finally, it calls `showAvatarsOfSelectedContacts` to potentially update the assigned contacts avatar list.

Parameters:

Name	Type	Description
`i`	number	The index of the contact in the list.

Source:

addTask_part1.js, line 384

`unclickCreatedContact(toggleIndex) → {void}`

This function removes the visual highlight from the previously highlighted contact (for screens wider than 800px). It uses the `toggleIndex` variable to identify the contact that was previously highlighted. It removes the "contactClicked" class from the button and resets the email color to its original state.

Parameters:

Name	Type	Description
`toggleIndex`	number	The index of the previously highlighted contact (should be the same value stored in the `toggleIndex` variable).

Source:

contacts_part2.js, line 282

Returns:

Type

void

`updateCards()`

Updates all card sections on the board by calling individual update functions for each section ("todo", "progress", etc.).

Source:

board_part1.js, line 76

(async) `updateContact(index)` → {Promise.<void>}

This function updates an existing contact at the specified index. It checks if all required fields (name, email, phone) are filled before updating.

Parameters:

Name	Type	Description
`index`	number	The index of the contact to update in the `localContacts` array.

Source:

contacts_part2.js, line 387

Returns:

A promise that resolves when the update process is complete.
Type

Promise.<void>

`updateDueDateContainers(overdueCards, upcomingCards)`

Updates UI with overdue/upcoming due date or "No upcoming due dates found" message. This function checks for overdue cards: - If overdue cards exist, it sets "Missed Deadline" text and gets the oldest overdue date. - If no overdue cards exist, it checks for upcoming cards: - If upcoming cards exist, it formats the closest upcoming due date. - Otherwise, it sets the message to "No upcoming due dates found". Finally, it calls functions to update the UI elements with the formatted output and deadline text/button styling.

Parameters:

Name	Type	Description
`overdueCards`	array	Array of overdue card objects.
`upcomingCards`	array	Array of upcoming card objects.

Source:

summary.js, line 285

`(async) updateUserContactsInRemote() → {Promise.<void>}`

Updates the remote storage with the current list of user contacts.

Source:

contacts_part1.js, line 93

Returns:

A promise that resolves when the remote storage is updated.
Type

Promise.<void>

`(async) updateUserContactsInRemoteAfterRegistration() → {Promise.<void>}`

Updates the remote storage with the current list of user contacts.

Source:

contacts_part1.js, line 84

Returns:

A promise that resolves when the remote storage is updated.
Type

Promise.<void>

`urgentButtonColor(urgentButtonClass)`

Updates the ".urgent-button" element class list for urgency (if it exists). This function retrieves the element with class ".urgent-button". - It removes the "missed-deadline" class (if present). - If `urgentButtonClass` is provided, it adds that class to the element.

Parameters:

Name	Type	Description
`urgentButtonClass`	string	Class name to add for urgency indication.

Source:

summary.js, line 354

`urgentNumber()`

Calculates and displays the number of tasks marked as "Urgent" based on the 'priority.urgency' property within each card object. Performs error handling if the container element with ID 'urgent-number' is not found.

Source:

summary.js, line 155

`urgentPriorityButtonStylingWhenClicked(button)`

This function applies styling for the clicked urgent priority button, highlighting it and resetting styles for other priority buttons.

Parameters:

Name	Type	Description
`button`	Element	The clicked urgent priority button element.

Source:

addTask_part1.js, line 130

`validateConfirmedPassword() → {boolean}`

Validates the input to confirm the password entered by the user.

Source:

signup.js, line 47

Returns:

Returns true if the confirmed password matches the original password, otherwise returns false.
Type

boolean

`validateEmailAddress() → {boolean}`

Validates the user's email address.

Source:

signup.js, line 5

Returns:

Returns true if the email address is valid, otherwise false.
Type

boolean

`validateFormFields(email, password, name) → {boolean}`

Validates form fields and displays an alert if any field is empty.

Parameters:

Name	Type	Description
`email`	string	User's email address.
`password`	string	User's password.
`name`	string	User's name.

Source:

signup.js, line 206

Returns:

- Returns true if all fields are filled, otherwise false.
Type

boolean

`validateTaskData()`

This function validates the provided task data. It checks for missing title, due date, or invalid category. If any are missing, it displays corresponding error messages and returns false. Otherwise, it returns true.

Source:

addTask_part2.js, line 338

`wrongPasswordMessage()`

Displays an error message when the password is entered incorrectly.

Source:

login.js, line 36

Type Definitions

`CreatedSubtask`

An array containing objects representing created subtasks. Each object has properties for "text" (the subtask description) and "done" (a boolean indicating completion status).

Type:

- object

Properties:

Name	Type	Description
`text`	string	The description of the subtask.
`done`	boolean	Whether the subtask is marked as completed.

Source:

addTask_part2.js, line 106

`TaskCategory`

An array containing task category objects. Each object has properties for name and category color.

Type:

- object

Properties:

Name	Type	Description
`name`	string	The name of the task category.
`categoryColor`	string	The color of the task category represented in RGB format.

Source:

addTask_part2.js, line 1

Documentation generated by JSDoc 4.0.3 on Tue May 07 2024 10:05:37 GMT+0200 (Mitteleuropäische Sommerzeit)