**AuthService/ User Screen:**

**Potential Screens for Each Role**

**1. Admin**

Admins generally have the highest privileges, so they would have access to more screens. The getLoggedInUserDetails method can be used in the following ways:

1. **Admin Dashboard**
   * Displays a summary of admin's profile and high-level organizational data.
   * Uses getLoggedInUserDetails to show the admin's information, such as name and email.
2. **User Management Screen**
   * Allows admins to view, update, or delete user details.
   * The method could fetch the currently logged-in admin's details to enforce restrictions (e.g., ensuring admins cannot modify their own super-admin role).
3. **Activity Log/History Screen**
   * Shows actions performed by the admin.
   * Admin details can be used to filter logs for the logged-in user.
4. **Settings/Account Management**
   * Allows the admin to update personal details such as their name, email, or password.
   * Fetches the admin's details for display and updates.

**2. Manager**

Managers typically oversee a subset of employees and may have mid-level privileges. The getLoggedInUserDetails method could be used in:

1. **Manager Dashboard**
   * Displays the manager's profile and a summary of team performance.
   * Uses getLoggedInUserDetails to show the manager's details.
2. **Team Overview Screen**
   * Allows managers to view and manage their team's information.
   * Uses the method to identify the logged-in manager for role-based access.
3. **Attendance Management Screen**
   * Shows the attendance records of team members.
   * Can display the manager's profile alongside team information.
4. **Report Generation Screen**
   * Allows managers to generate performance reports.
   * Uses the logged-in manager's details to personalize reports or restrict access to their team's data.
5. **Account Management**
   * Similar to the admin, this screen allows managers to view and update their personal account details.

**3. Employee**

Employees have the most limited access but still need several screens for their day-to-day activities. The getLoggedInUserDetails method can be used in:

1. **Employee Dashboard**
   * Displays the employee's profile and daily tasks.
   * Uses the method to show the logged-in employee's details.
2. **Attendance Screen**
   * Allows employees to clock in and out or view attendance records.
   * Shows the logged-in user's details for personalized attendance data.
3. **Profile Screen**
   * Displays and allows updates to the employee's personal details.
   * Relies heavily on getLoggedInUserDetails for fetching and displaying user data.
4. **Leave Application Screen**
   * Allows employees to apply for leave.
   * Uses the logged-in employee's details to pre-fill the application form.
5. **Payslip/Salary History Screen**
   * Displays salary details and history.
   * The employee's profile fetched by the method can be used to display personalized data.

**Screen Summary**

| **Role** | **Potential Screens** |
| --- | --- |
| **Admin** | Dashboard, User Management, Activity Log, Settings/Account Management |
| **Manager** | Dashboard, Team Overview, Attendance Management, Report Generation, Settings/Account Management |
| **Employee** | Dashboard, Attendance, Profile, Leave Application, Payslip/Salary History |

**Shared Screens Across Roles**

In some cases, the same screen can serve multiple roles but with role-specific data:

1. **Dashboard**: Different content for Admin, Manager, and Employee.
2. **Profile/Settings Screen**: Same layout but populated with role-specific details.
3. **Activity Log**: Admin sees all actions; Managers see team logs; Employees see personal logs.

**AdvanceSalary Screen:**

**Screens for Each Role**

**1. Employee Role**

Employees will primarily interact with the system to apply for advance salary, view pending requests, and check the status of their advance salary requests. They will need the following screens:

1. **Advance Salary Application Screen**
   * Allows the employee to apply for an advance salary by entering the amount, reason, and date.
   * **API used**: createAdvanceSalary
2. **Pending Applications Screen**
   * Displays a list of advance salary requests that are still pending approval.
   * **API used**: getAdvanceSalaries (to retrieve all advance salary records, and filter by status in the frontend)
3. **Advance Salary History Screen**
   * Displays a history of all the employee’s advance salary applications, including those that have been approved or rejected.
   * **API used**: getAdvanceSalaries (fetch all salary records for the employee, filter by approval status)
4. **Advance Salary Status Screen**
   * Displays detailed information on a specific advance salary request (approved, rejected, or pending).
   * **API used**: getAdvanceSalaryById
5. **Advance Salary Date Range Filter Screen** (Optional)
   * Allows employees to filter advance salary requests by a specific date range.
   * **API used**: getAdvanceSalariesByDateRange

**2. Manager Role**

Managers typically review and approve or reject employee advance salary applications. Therefore, they will need the following screens:

1. **Advance Salary Review Screen**
   * A list of all pending advance salary requests from employees. Managers can approve or reject these requests.
   * **API used**: getAdvanceSalaries (to view all pending requests)
2. **Advance Salary Details Screen**
   * Displays detailed information about a specific advance salary request, including the employee's reason, amount, and date.
   * **API used**: getAdvanceSalaryById
3. **Advance Salary Approval Screen** (can be part of the details screen or separate)
   * Allows the manager to approve or reject the advance salary application.
   * **API used**: updateAdvanceSalary (to update the status after approval/rejection)
4. **Advance Salary Date Range Filter Screen** (Optional)
   * Allows managers to view requests by a date range.
   * **API used**: getAdvanceSalariesByDateRange

**3. Admin Role**

Admins typically have full control over the system and can view, approve, or reject any requests. They can also manage employees' and managers' records. Therefore, admins will need the following screens:

1. **Advance Salary Overview Screen**
   * A comprehensive view of all advance salary records across the system. This can include both pending, approved, and rejected requests.
   * **API used**: getAdvanceSalaries (to fetch all requests)
2. **Advance Salary Review Screen**
   * Similar to the manager, the admin can review and approve or reject advance salary requests from all employees.
   * **API used**: getAdvanceSalaries (to view all pending requests)
3. **Advance Salary Details Screen**
   * Displays detailed information about a specific advance salary request.
   * **API used**: getAdvanceSalaryById
4. **Advance Salary Date Range Filter Screen** (Optional)
   * Allows the admin to filter advance salary requests by a specific date range.
   * **API used**: getAdvanceSalariesByDateRange
5. **Admin Management Screen** (for managing employee and manager records)
   * Admins can manage the roles of employees and managers who apply for advance salary.
   * **API used**: createAdvanceSalary, updateAdvanceSalary, deleteAdvanceSalary (for managing records)

**Summary of Screens**

| **Role** | **Possible Screens** |
| --- | --- |
| **Employee** | 1. Advance Salary Application Screen  2. Pending Applications Screen  3. Advance Salary History Screen  4. Advance Salary Status Screen  5. Advance Salary Date Range Filter Screen (Optional) |
| **Manager** | 1. Advance Salary Review Screen  2. Advance Salary Details Screen  3. Advance Salary Approval Screen  4. Advance Salary Date Range Filter Screen (Optional) |
| **Admin** | 1. Advance Salary Overview Screen  2. Advance Salary Review Screen  3. Advance Salary Details Screen  4. Advance Salary Date Range Filter Screen (Optional)  5. Admin Management Screen |

**How Many Screens Should You Create?**

Given the different features for each role, you may need **10 to 12 screens** in total:

* **Employee Screens**: 4 to 5 screens.
* **Manager Screens**: 3 to 4 screens.
* **Admin Screens**: 4 to 5 screens.

**Attendance And other API:**

 Attendance Check-In and Check-Out

 View Attendance History

 Overtime Records (if needed)

**Bonus Entity Screen:**

**1. Bonus Dashboard Screen**

A central screen where users can access their bonus-related information, such as the latest bonus, total bonus, and the option to apply for bonuses or view bonus history.

* **Main Features**:
  + Display the total bonus for the user.
  + Show the latest bonus details.
  + A button to navigate to the bonus history or apply for a bonus.

**2. Bonus History Screen**

This screen allows users to view their bonus history, including the bonus amount and date for each bonus received.

* **Main Features**:
  + Display a list of bonuses for a user (using getBonusesBetweenDates).
  + Include filters to search bonuses within a specific date range.
  + Show bonus amount and bonus date.
  + Option to view details or perform other actions like requesting bonuses.

**3. Apply for Bonus Screen (Optional)**

This screen could allow admins or managers to manually assign or create bonuses for users. This might include the ability to define bonus amounts, user, and bonus date, or calculate bonuses based on certain conditions.

* **Main Features**:
  + Form to create a bonus for a user (using createBonus).
  + Input fields for bonus amount, user, and date.
  + Submit button to create the bonus and save it in the database.

**4. Bonus Details Screen**

This screen could be used to view detailed information about a specific bonus, such as the bonus amount, user details, and the bonus date.

* **Main Features**:
  + Display detailed information for a specific bonus (using getLatestBonusForUser).
  + View the bonus amount, bonus date, and user info.
  + Option to update the bonus details if necessary.

**5. Leave Bonus Deduction Screen**

This screen is designed to show the calculation for leave-related bonus deductions, using the API to calculate bonus deductions based on the number of leave days.

* **Main Features**:
  + Input for leave days.
  + Display the calculated bonus deduction for the given leave days.
  + Option to apply or update deductions based on leave days.

**1. Branch List Screen**

* **Purpose**: To display all branches in a list format, potentially with pagination or search functionality.
* **Operations**: Fetch and display all branches using the getAllBranches() method.
* **Features**:
  + Show branch details such as name, address, email, phone, etc.
  + Option to view details of each branch.
  + Option to search branches by address or name.
  + Option to delete or update a branch.

**2. Branch Detail Screen**

* **Purpose**: To show detailed information for a selected branch.
* **Operations**: Fetch branch data by ID using getBranchById().
* **Features**:
  + Display full details of the branch (name, address, email, cell, created/updated timestamps).
  + Option to edit branch details (navigate to Branch Edit Screen).
  + Option to delete the branch.

**3. Create Branch Screen**

* **Purpose**: To allow users to create a new branch by providing the required details and optionally uploading a branch photo.
* **Operations**: Use the createBranch() method to create a new branch.
* **Features**:
  + Form for entering branch name, address, email, cell, and an optional branch photo.
  + Option to submit the form to create the branch.

**4. Update Branch Screen**

* **Purpose**: To update details of an existing branch.
* **Operations**: Use the updateBranch() method with the branch's ID to update branch details.
* **Features**:
  + Display current branch details in an editable form.
  + Option to upload a new branch photo.
  + Submit button to save the updates.

**5. Branch Search Screen**

* **Purpose**: To search for branches based on different criteria, such as branch name, address, email, or cell.
* **Operations**:
  + Use the getBranchByName(), getBranchesByAddressKeyword(), getBranchByEmail(), and getBranchByCell() methods to filter branches.
* **Features**:
  + Input fields for searching by branch name, address, email, or cell.
  + Display search results dynamically as the user enters keywords.

**6. Branch Count Screen**

* **Purpose**: To show the total number of branches in the system.
* **Operations**: Use the countBranches() method to get the total count.
* **Features**:
  + Display the number of branches currently in the system.

**7. Branch Image Preview Screen**

* **Purpose**: To preview the branch image (if uploaded) in full size or manage it.
* **Operations**: Display the branch image stored in the backend, if available.
* **Features**:
  + Option to view or update the branch image.
  + Preview of the image uploaded for a specific branch.

**8. Delete Branch Confirmation Screen**

* **Purpose**: To confirm before deleting a branch.
* **Operations**: Call deleteBranch() to delete the branch upon user confirmation.
* **Features**:
  + Display a confirmation prompt before deletion.
  + Option to cancel or confirm the deletion.

**Department Screen:**

**Department Overview Screen**

* **Features:**
  + Display a list of all departments.
  + Include department names, email addresses, employee counts, and branch details.
  + Add a button to view department details, update, or delete.

**API used:**

* + GET /api/departments/all

**2. Department Details Screen**

* **Features:**
  + Display detailed information about a specific department, including its photo, branch, and employee count.
  + Provide navigation options for editing or deleting the department.

**API used:**

* + GET /api/departments/find/{id}

**3. Create Department Screen**

* **Features:**
  + A form to input department details (e.g., name, email, cell, employee number, branch).
  + Upload a department photo (optional).
  + Submit the form to create a new department.

**API used:**

* + POST /api/departments/create

**4. Update Department Screen**

* **Features:**
  + A pre-filled form with the existing department details.
  + Allow editing fields such as name, email, cell, employee number, and branch.
  + Option to update or remove the department photo.
  + Save changes.

**API used:**

* + PUT /api/departments/update/{id}

**5. Search Departments Screen**

* **Features:**
  + Input fields for various search parameters like:
    - **Department Name:** GET /api/departments/findByName/{name}
    - **Branch ID:** GET /api/departments/findByBranch/{branchId}
    - **Branch Name:** GET /api/departments/findByBranchName/{branchName}
    - **Email:** GET /api/departments/findByEmail/{email}
    - **Employee Count Range:** GET /api/departments/findByMinEmployeeCount/{minEmployeeCount}, GET /api/departments/findByMaxEmployeeCount/{maxEmployeeCount}
  + Display search results in a list with action buttons for details, update, or delete.

**6. Department Analytics Screen**

* **Features:**
  + Display statistics such as:
    - Number of departments by branch: GET /api/departments/countByBranch/{branchId}
    - Departments with no employees: GET /api/departments/findWithNoEmployees
    - Departments updated after a specific date: GET /api/departments/findUpdatedAfter
    - Departments created within a date range: GET /api/departments/findByCreatedAtRange
  + Use charts or graphs to visualize data.

**7. Department Deletion Confirmation Screen**

* **Features:**
  + Confirmation dialog or screen asking the user to confirm the deletion of a department.
  + Show department details for context.
  + Execute deletion on confirmation.

**API used:**

* + DELETE /api/departments/delete/{id}

**8. Department Photo Management Screen**

* **Features:**
  + Upload or update a department photo.
  + Preview the current photo or upload a new one.
  + Save changes.

**API used:**

* + POST /api/departments/create (for creation)
  + PUT /api/departments/update/{id} (for updates)

**9. Departments Without Employees Screen**

* **Features:**
  + Display a list of all departments with no employees.
  + Allow adding employees to the department or viewing branch details.

**API used:**

* + GET /api/departments/findWithNoEmployees

**10. Department Creation Analytics Screen**

* **Features:**
  + Show departments created within a specific date range.
  + Include filters for start and end dates.
  + Present results in a list or visualized format (e.g., bar chart).

**API used:**

* + GET /api/departments/findByCreatedAtRange

**11. Branch-Specific Departments Screen**

* **Features:**
  + Display departments for a specific branch.
  + Include branch name or ID as a filter.

**API used:**

* + GET /api/departments/findByBranch/{branchId}
  + GET /api/departments/findByBranchName/{branchName}

**12. Department Search by Employee Count Screen**

* **Features:**
  + Filter departments by employee count.
  + Include fields for minimum and maximum values.

**API used:**

* + GET /api/departments/findByMinEmployeeCount/{minEmployeeCount}
  + GET /api/departments/findByMaxEmployeeCount/{maxEmployeeCount}

**Leave Requests Overview Screen:**

**1. Leave Requests Overview Screen**

* **Features:**
  + Display a list of all leave requests.
  + Show leave details like start date, end date, leave type, status, and user name.
  + Include filters for **status** (pending, approved, rejected).
  + Include action buttons to view details, approve, or reject requests.

**APIs used:**

* + GET /api/leaves/pending (for pending requests).
  + GET /api/leaves/rejected (for rejected requests).
  + GET /api/leaves/user/{userId}/approved (for approved requests by user).

**2. Leave Details Screen**

* **Features:**
  + Show detailed information about a specific leave request, including:
    - Leave type, reason, request date, start/end dates, status, and remaining leaves.
    - User information (retrieved via User model).
  + Include action buttons for approving or rejecting the leave.

**APIs used:**

* + GET /api/leaves/find/{id}

**3. Create Leave Request Screen**

* **Features:**
  + A form to create a new leave request with fields:
    - Start date, end date, leave type (dropdown), reason (textarea).
    - Display remaining leave days.
  + Submit button to save the request.

**API used:**

* + POST /api/leaves/save

**4. Edit Leave Request Screen**

* **Features:**
  + A pre-filled form with the existing leave request details.
  + Allow editing fields like start date, end date, leave type, and reason.
  + Save updated leave request.

**API used:**

* + PUT /api/leaves/update/{id}

**5. My Leave Requests Screen**

* **Features:**
  + Display a list of leave requests created by the logged-in user.
  + Filter leave requests by **status** (pending, approved, rejected) or **leave type** (sick, unpaid, reserve).
  + Include options to view details, edit, or delete requests.

**APIs used:**

* + GET /api/leaves/user/{userId}/approved
  + GET /api/leaves/user/{userId}/range

**6. Approve/Reject Leave Requests Screen**

* **Features:**
  + Display a list of pending leave requests.
  + Include action buttons for **Approve** and **Reject**.
  + Show basic leave details like dates, type, and reason.

**APIs used:**

* + POST /api/leaves/approve/{id}
  + POST /api/leaves/reject/{id}

**7. Search Leave Requests Screen**

* **Features:**
  + Search leave requests based on:
    - **Leave Type**: GET /api/leaves/type/{leaveType}
    - **Reason**: GET /api/leaves/reason/{reason}
    - **Date Range**: GET /api/leaves/user/{userId}/range?startDate={startDate}&endDate={endDate}
  + Display the search results in a list.

**8. Leave Requests Analytics Screen**

* **Features:**
  + Visualize leave request data, such as:
    - Number of leave requests by **status** (pending, approved, rejected).
    - Leave types used most frequently.
    - Average leave duration per user.
  + Show analytics as charts or graphs.

**APIs used:**

* + Combination of GET APIs for pending, approved, rejected, and type-based leaves.

**9. Delete Leave Request Confirmation Screen**

* **Features:**
  + Show a confirmation dialog when deleting a leave request.
  + Display leave details for context before confirming the deletion.

**API used:**

* + DELETE /api/leaves/delete/{id}

**10. Filter Leaves by Type Screen**

* **Features:**
  + Display a dropdown or list to filter leaves based on their type (**sick, unpaid, reserve**).
  + Show results in a paginated or scrollable list.

**API used:**

* + GET /api/leaves/type/{leaveType}

**11. View Leaves by Date Range Screen**

* **Features:**
  + A date picker to select start and end dates.
  + Display all leave requests for the logged-in user within the selected range.

**API used:**

* + GET /api/leaves/user/{userId}/range?startDate={startDate}&endDate={endDate}

**12. Employee Leave Summary Screen**

* **Features:**
  + Show a summary of total, remaining, and used leaves for each employee.
  + Include filters to view data for a specific employee.

**API used:**

* + Aggregated data from GET APIs like approved leaves, pending leaves, and rejected leaves.

**Salary:**

**1. Salary Overview**

* **Purpose**: Display a summary of salaries for all employees.
* **Features**:
  + List all salaries with essential details (e.g., payment date, net salary, allowances).
  + Pagination or search/filter options (e.g., by user or date range).

**2. Salary Details**

* **Purpose**: View detailed information about a specific salary record.
* **Features**:
  + Breakdown of all components (medicare, tax, allowances, bonuses, overtime).
  + Associated user details and leaves impacting the salary.

**3. Create/Update Salary**

* **Purpose**: Allow admin or manager to create or modify salary records.
* **Features**:
  + Form for entering/updating salary details.
  + Validation for required fields (e.g., allowances, tax, net salary).
  + Option to attach related data like bonuses, leaves, or overtime.

**4. Salaries by Date Range**

* **Purpose**: View salaries processed within a specific date range.
* **Features**:
  + Date range picker for filtering records.
  + Display salary records in a tabular or list format.

**5. Latest Salary for Employee**

* **Purpose**: Show the latest salary record for a specific user.
* **Features**:
  + Search or select an employee by ID.
  + Display the most recent salary details.

**6. All Salaries**

* **Purpose**: Comprehensive list of all salaries in the system.
* **Features**:
  + Sorting and filtering options (e.g., by employee, payment date, or net salary).
  + Export functionality for records (e.g., to PDF or CSV).

**7. Delete Salary Record**

* **Purpose**: Manage deletion of salary records.
* **Features**:
  + Confirmation dialog before deletion.
  + Success or failure message after the operation.

**8. Employee-Specific Salary Records**

* **Purpose**: Allow employees to view their own salary history.
* **Features**:
  + Restricted view based on user authentication.
  + Display only the logged-in user's salary records.

**9. Salary Insights and Reports**

* **Purpose**: Provide analytical insights on salaries.
* **Features**:
  + Visualize data through graphs (e.g., total salaries over months, tax deductions).
  + Highlight trends or anomalies in salary distribution.

**10. Pending Overtime and Bonus Integration**

* **Purpose**: Provide a screen for managing overtime and bonuses for salary adjustments.
* **Features**:
  + Display pending overtime hours and bonuses for inclusion in salaries.
  + Allow adjustments before salary finalization.

**Additional Screens for Admin Management**

**11. User-Specific Salary Management**

* Purpose: Manage and view all salary records related to a specific user.

**12. Approval Requests for Salary Adjustments**

* Purpose: Approve or reject salary-related changes (e.g., bonuses or deductions).

**PaySlipService**

**1. Employee Payslip History**

* **Purpose**: Allows employees to view their own payslip history.
* **Features**:
  + List of payslips for the logged-in employee.
  + Search/filter by billing date, payment method, or status.
  + View details of individual payslips.

**2. Payslip Details**

* **Purpose**: Display detailed information about a specific payslip.
* **Features**:
  + Total amount, billing date, payment method, and status.
  + Associated salary details (allowances, deductions).
  + Paid by and received by user information.

**3. Manager Payslip View**

* **Purpose**: Managers can view payslips for their supervised employees.
* **Features**:
  + List of payslips issued to employees under the manager.
  + Filter by employee, billing date, or status.
  + View payslip details for verification or reporting.

**4. Payslips by Status**

* **Purpose**: View payslips filtered by their status (e.g., Paid, Pending, Canceled).
* **Features**:
  + Status filter dropdown (or tabs for each status).
  + List of payslips with a status indicator.

**5. Payslips by Payment Method**

* **Purpose**: View payslips based on the payment method (e.g., Bank Transfer, Cash).
* **Features**:
  + Filter payslips by payment method.
  + List of payslips showing the chosen payment method.

**6. Payslip Creation**

* **Purpose**: Allow admin or managers to create new payslips.
* **Features**:
  + Form for entering required details (employee ID, salary ID, total amount, payment method).
  + Validation for required fields.
  + Confirmation message upon successful creation.

**7. Payslips by Billing Date Range**

* **Purpose**: View payslips generated within a specific date range.
* **Features**:
  + Date range picker to filter records.
  + Display payslips in a table or list format.

**8. Payslips for a Specific Salary**

* **Purpose**: Show the payslip associated with a specific salary record.
* **Features**:
  + Search by salary ID to fetch the corresponding payslip.
  + Option to navigate from salary details to its associated payslip.

**9. Payslip Count by Status**

* **Purpose**: Provide a summary count of payslips by status.
* **Features**:
  + Display counts of Paid, Pending, and Canceled payslips.
  + Use graphs or charts for better visualization.

**10. Employee and Status-Specific Payslips**

* **Purpose**: Show payslips for a specific employee filtered by status.
* **Features**:
  + Combine filters for employee and status.
  + List or card view of filtered payslips.

**11. Dashboard with Payslip Insights**

* **Purpose**: A summary screen providing analytics and insights for payslip management.
* **Features**:
  + Charts for payslips by status, date, and payment method.
  + Quick navigation links to detailed views.

**12. Admin/Manager Payslip Approval**

* **Purpose**: Review and approve/reject pending payslips (if applicable).
* **Features**:
  + List of pending payslips requiring action.
  + Approve/reject buttons with a status update.