React Native Hometask — Payslips (iOS & Android)

Build a small React Native application (TypeScript) for managing and viewing payslips. Use mock data only. Focus on native file handling, clean architecture, accessibility, and a polished mobile UX.

Requirements

Payslip Model

Each payslip contains:

- **id:** unique identifier
- **fromDate:** start date of the covered period (ISO format suggested)
- **toDate:** end date of the covered period (ISO format suggested)
- **file:** a bundled asset representing the payslip (PDF or image). You may reuse the same file across items.

Screens

1. **Payslip List**
    - Display a scrollable list of payslips.
    - Each list item shows the period: “fromDate – toDate”.
    - Tapping an item navigates to the details screen.
    - Additional requirements:
        - Allow the user to **sort** payslips by:
            - Most recent first
            - Oldest first
        - (Nice to have) Provide a simple **filter** (e.g., by year or text search).

2. **Payslip Details**
    - Display **id**, **fromDate**, **toDate**, and an indicator of the file type (PDF/image).
    - Provide a **Download Payslip** action:
        - Saves the associated file to device storage and confirms success/failure to the user (e.g., show an alert).
    - (Nice to have) Provide an **Open/Preview** action to view the file using a native viewer.

Data & State

- Initialize the app with several payslips in **in-memory state (React Context, Redux etc**)
- No API calls or server is required.

Native File Handling (Download)

Implement the “download” by copying the bundled file to a user-accessible or app-specific directory and informing the user of the saved location.

Acceptable approaches:

- Use a well-supported React Native library for filesystem operations and handle Android runtime permissions where applicable.
- (Nice to have) Opening/previewing the saved file is a plus.

Testing

We don’t expect full coverage, but we do want to see that you can write a couple of meaningful tests.

- Add at least 2–3 tests, for example:
- A unit test for a date formatting helper.A test that renders the Payslip List and checks that items appear as expected.
- You may use Jest and any common testing library (e.g. Testing Library for React Native).

Platform Scope

- **React Native (CLI) or Expo**, **TypeScript**, platforms **iOS** and **Android**.

Suggested Project Structure (Informational)

- Separate concerns (screens, navigation, components, state/store, utilities).
- Keep components small and composable.
- Include basic date formatting utilities and a simple theme.

Use of AI Tools

You are allowed to use AI-assisted tools (ChatGPT, Copilot, etc.), but:

- You are expected to **understand and own all the code** you submit.
- During the review, we may ask you to:
    - Walk through selected files or components.
    - Explain why certain patterns or libraries were used.
    - Discuss trade-offs and possible improvements.
- Please avoid copy-pasting code you can't explain or debug.

README & Submission

A concise **README** including:

- Tech stack and choices (RN CLI/Expo, libraries used).
- How to run the app on **iOS** and **Android** (commands, prerequisites).
- Known limitations and what you would improve with more time.

Repository:

- Public Git repository.
- Commit history doesn't have to be perfect, but avoid a single "big dump" commit if possible.

Evaluation Criteria

Reviewers will assess:

Architecture & Code Quality

- Clear separation of concerns (screens, components, navigation, state, utilities, theme).
- Idiomatic React Native and TypeScript usage.
- Readable, maintainable code (naming, structure, minimal duplication).
- Sensible state management choice and usage (Context/Redux/Zustand/etc).

Type Safety & Tooling

- Appropriate types and interfaces across the app (including navigation params).
- Minimal or well-justified use of `any`.
- Basic linting/formatting in place and followed (e.g. ESLint/Prettier).

Native Handling & Platform Awareness

- Correct file save logic using a filesystem solution.
- Proper handling of Android runtime permissions where applicable.
- Clear, actionable user feedback on success/failure of downloads.
- (Bonus) Opening/previewing files and handling platform-specific differences gracefully.

UX, Accessibility & Polish

- Clean, consistent layout and visual hierarchy.
- List and details screens are intuitive to navigate.

Stability & Error Handling

- No obvious crashes or red screens under normal usage.
- Reasonable handling of error cases (permissions denied, file write failures, etc.).
- Avoids noisy console errors and obvious unhandled promise rejections.

Testing

- Presence of at least a couple of meaningful tests (e.g. utilities, basic UI behavior).
- Tests are focused and readable, not just snapshots for the sake of it.

Documentation & Communication

- A concise README with accurate setup and run instructions for iOS and Android.
- Short explanation of architecture, library choices, and trade-offs.
- Notes on limitations and what they would improve with more time.