**Product Requirements Document: Productivity App**

**1. Elevator Pitch**

This productivity app is a minimalist, powerful tool designed to help users manage and complete important projects effortlessly. It features a summary page of all open projects sorted by user-defined categories, detailed project pages with sourced notes and dependency-driven to-do lists, and specialized views for daily tasks and outstanding items. With offline access via local storage and multi-device sync, it’s built for personal productivity with the potential to scale for broader audiences like freelancers or teams.

**2. Who is this app for**

This app is initially designed for the product owner—an individual who needs a tailored solution to track and complete projects efficiently. While focused on personal use, its functionality could later appeal to broader audiences such as freelancers, students, or small teams seeking a straightforward project management tool.

**3. Functional Requirements**

* **Project Summary Page**: Display all open projects, sorted by user-defined categories (e.g., Work, Personal, Hobbies).
* **Detailed Project Page**: For each project, include:
  + **Captured Notes**: A text field for notes with fields for source (e.g., meeting, phone call, text message, email) and contact (e.g., name or identifier).
  + **To-Do List**: Editable, checkable tasks with dependency markers (e.g., Task B requires Task A to be completed first).
  + **Project Notes**: A dedicated space for general project-specific notes.
* **Due Today Page**: A daily overview showing:
  + To-dos due today, separated into:
    - Tasks requiring contact with others (linked to a named contact).
    - Tasks completable independently.
    - Items flagged for today’s meetings.
  + A filter to select any contact and display all outstanding to-dos (across all projects) where that contact is listed, regardless of due date.
* **Outstanding To-Dos Page**: A list of all incomplete to-dos not due today, sorted by whether they’re flagged (e.g., high-priority markers).
* **Data Storage**: Store all data locally on the device for offline access.
* **Sync Capability**: Enable syncing across multiple devices when online.
* **Minimalist Design**: Simple, uncluttered interface prioritizing usability.

**4. User Stories**

* **As a user**, I want a summary page of my projects sorted by categories I define, so I can quickly see what needs focus.
* **As a user**, I want to view a project’s details, including notes and to-dos with dependencies, so I can manage it effectively.
* **As a user**, I want to log notes with sources and contacts, so I can track where information came from.
* **As a user**, I want a "Due Today" page separating tasks by contact needs and meeting items, so I can plan my day efficiently.
* **As a user**, I want to filter to-dos by contact across all projects, so I can see everything I need to discuss with someone.
* **As a user**, I want a list of outstanding to-dos not due today, sorted by flags, so I can prioritize future work.
* **As a user**, I want to use the app offline and sync it across devices, so I’m productive anywhere.

**5. User Interface**

The app will feature a **minimalist design** with a clean, functional layout:

* **Summary Page**: A list or grid of projects grouped by user-defined categories, with simple titles and minimal visuals (e.g., monochrome icons).
* **Detailed Project Page**: A single-column layout with:
  + A header with the project name.
  + A "Captured Notes" section with collapsible entries (source and contact tagged).
  + An interactive "To-Do List" with checkboxes, edit buttons, and dependency indicators (e.g., arrows).
  + A "Project Notes" text box.
* **Due Today Page**: A segmented view with:
  + Three collapsible sections: "Contact Required," "Independent Tasks," and "Meeting Items," each listing to-dos due today.
  + A dropdown or search bar to filter by contact, showing all related outstanding to-dos.
* **Outstanding To-Dos Page**: A list with flagged items at the top (e.g., bolded or starred), followed by unflagged items.
* **Visual Style**: Black-and-white or muted tones, sans-serif fonts, and ample white space.
* **Navigation**: A bottom or side bar with icons for Summary, Due Today, Outstanding To-Dos, and back buttons within pages.

**User Interface Description Document: Productivity App ("Linear Flow" Design)**

**Layout Structure**

* **Summary Page**: A table layout with user-defined categories (e.g., "Work," "Personal") as column headers across the top row. Projects stack vertically under each category as compact cards. The table fits all categories on-screen when possible; horizontal scrolling reveals additional columns, and vertical scrolling per column handles extra projects.
* **Detailed Project Page**: A single-column, top-down layout with the project name at the top, followed by "Captured Notes," "To-Do List," and "Project Notes" sections.
* **Due Today Page**: A single-column view with three collapsible sections stacked vertically, topped by a sticky contact filter bar.
* **Outstanding To-Dos Page**: A single-column list with flagged items at the top, separated by a faint line from unflagged items.
* **Navigation**: A fixed bottom bar with icons for Summary, Due Today, and Outstanding To-Dos, plus a back arrow within pages.

**Core Components**

* **Summary Page Cards**: Each project card shows: [Project Name] | Remaining: [X] | Available: [Y] (X = total incomplete to-dos, Y = to-dos with no pending dependencies). Tappable to open the Detailed Project Page.
* **Captured Notes Section**: Expandable bubbles listing notes with tagged source (e.g., "Meeting") and contact (e.g., "John").
* **To-Do List**: Checkboxes for tasks, with faint arrows indicating dependencies. Editable via an inline text field on tap.
* **Project Notes**: A plain text box for general project notes, auto-saving on input.
* **Due Today Sections**: Three collapsible lists: "Contact Required" (tasks linked to a contact), "Independent Tasks" (no contact needed), and "Meeting Items" (flagged for today’s meetings). A sticky filter bar above shows all to-dos for a selected contact across projects.
* **Outstanding To-Dos List**: Flagged items bolded at the top, followed by unflagged items.

**Interaction Patterns**

* **Tapping**: Tap a project card on the Summary Page to open its Detailed Project Page. Tap a note bubble to expand/collapse it. Tap a to-do to edit or check it off.
* **Scrolling**: Horizontal scrolling on the Summary Page for additional categories; vertical scrolling per column for more projects. Vertical scrolling on other pages for content overflow.
* **Filtering**: Tap the contact filter bar on the Due Today Page, select a contact from a dropdown, and see all related to-dos update below.
* **Collapsing**: Tap section headers on the Due Today Page to collapse/expand lists.
* **Navigation**: Tap bottom bar icons to switch views; tap the back arrow to return to the previous page.

**Visual Design Elements & Color Scheme**

* **Style**: Minimalist with a black-and-white palette—black text/icons on a white background. Priority flags or dependency arrows in muted gray.
* **Icons**: Simple, monochrome (e.g., house for Summary, clock for Due Today, list for Outstanding To-Dos).
* **Spacing**: Ample white space between elements for clarity, with tight stacking in the Summary Page table to maximize visibility.
* **Dividers**: Faint gray lines separate table columns, Due Today sections, and flagged/unflagged to-dos.

**Mobile, Web App, Desktop Considerations**

* **Mobile**: Bottom navigation bar fits small screens; table columns shrink to 1-2 per view with horizontal scrolling. Vertical scrolling per column remains fluid.
* **Web App**: Wider screens show more table columns (3-5) without scrolling; navigation bar stays at the bottom for consistency. Mouse hover on cards shows a tooltip with full project name if truncated.
* **Desktop**: Table columns expand to 5-7, with a pinned navigation bar. Detailed Project Page sections can optionally split into two columns (e.g., To-Do List beside Notes) if screen width allows.

**Typography**

* **Font**: Sans-serif (e.g., Roboto or similar) for readability.
* **Sizes**: Project names and headers at 16px, to-do text and notes at 14px, source/contact tags at 12px—all in regular weight. Flagged items bolded.
* **Consistency**: Uniform font across all pages, with minimal styling (bold for emphasis only).

**Accessibility**

* **Contrast**: High contrast (black on white) meets WCAG 2.1 AA standards.
* **Text Size**: Adjustable via device settings; minimum 14px ensures legibility.
* **Navigation**: Bottom bar icons include text labels on long-press or screen reader activation.
* **Interaction**: Tap targets (e.g., cards, checkboxes) are at least 48x48px for touch accessibility.
* **Screen Readers**: Table headers announced as categories, project cards as "Project [Name], Remaining [X], Available [Y]," and to-dos with dependency status.

**Software Requirements Specification Document: Productivity App**

**System Design**

* **Overview**: A mobile-first productivity app with offline functionality and optional multi-device sync.
* **Components**:
  + Mobile app (initially Android/iOS) with local storage for offline use.
  + Optional cloud sync service for future multi-device support.
* **Deployment**: Standalone mobile app with local data; cloud backend optional for sync.

**Architecture Pattern**

* **Pattern**: MVVM (Model-View-ViewModel).
  + **Model**: Manages data (projects, tasks, notes) and storage logic.
  + **View**: Displays the UI (e.g., Summary Page, Due Today Page).
  + **ViewModel**: Connects data to the UI and handles user actions (e.g., adding a task).
* **Reasoning**: Simple to understand, supports offline use, and allows reactive UI updates.

**State Management**

* **Approach**: Centralized state with reactive updates.
* **Implementation**:
  + In-memory state for active data (e.g., current project being viewed).
  + Local database (SQLite) for persistent storage.
  + Sync logic to update data across devices when online (future phase).
* **Tool**: Provider (built into Flutter) for simplicity and ease of learning.

**Data Flow**

* **Flow**:
  1. User interacts with the app (e.g., checks off a task).
  2. ViewModel updates the in-memory state and saves to local database.
  3. UI refreshes automatically via reactive bindings.
  4. When online (future), sync service pushes/pulls changes to/from a backend.
* **Offline Handling**: All actions save locally; sync happens in the background when connected.

**Technical Stack**

* **Frontend**:
  + **Framework**: Flutter (cross-platform, beginner-friendly, mobile-first).
  + **Language**: Dart (easy syntax, comes with Flutter).
* **Local Storage**: SQLite via the sqflite package (simple, built for Flutter).
* **Backend (Sync, Future)**: Firebase (no-code setup for beginners, handles sync and authentication).
* **Tools**:
  + **IDE**: Visual Studio Code (free, beginner-friendly) with Flutter/Dart plugins.
  + **Build**: Flutter CLI (included with Flutter installation).
  + **Version Control**: Git (optional, can be learned later).
* **Learning Resources**: Flutter’s official docs (flutter.dev), free YouTube tutorials.

**Authentication Process**

* **Initial Phase**: No authentication (local app for personal use).
* **Future Sync**:
  + Firebase Authentication with email/password login.
  + Steps:
    1. User signs up/logs in via Firebase UI (pre-built).
    2. Firebase issues a token stored locally.
    3. Token sent with sync requests to secure data.

**Route Design**

* **Screens** (Flutter uses “routes” for navigation):
  + /summary: Project Summary Page.
  + /project/:id: Detailed Project Page (dynamic ID for each project).
  + /due-today: Due Today Page.
  + /outstanding: Outstanding To-Dos Page.
* **Navigation**: Bottom navigation bar switches between screens; back button within pages.

**API Design**

* **Note**: Only for future sync with Firebase (optional).
* **Approach**: Use Firebase Firestore (NoSQL database) with built-in sync.
* **Operations**:
  + **Write**: Local changes (e.g., new task) saved to Firestore when online.
  + **Read**: Pull updates from Firestore to local SQLite when connected.
* **Format**: Handled by Firebase SDK (no manual API coding needed).

**Database Design ERD**

* **Entities** (stored in SQLite locally):
  + **Project**: id (PK), name, category, createdAt, updatedAt
  + **Task**: id (PK), projectId (FK), description, dueDate, isCompleted, dependsOnTaskId (FK, nullable), contact (nullable), isFlagged, isMeetingItem
  + **Note**: id (PK), projectId (FK), content, source, contact, createdAt
* **Relationships**:
  + Project 1:N Task
  + Project 1:N Note
  + Task 1:1 Task (self-referential for dependencies)
* **Future Sync**: Mirrored in Firebase Firestore as collections: projects, tasks, notes.

**Next Steps for You**

Since you’re new to programming, here’s a simple plan to get started:

1. **Install Tools**:
   * Download Flutter (flutter.dev) and Visual Studio Code.
   * Follow Flutter’s “Get Started” guide to set up your environment.
2. **Learn Basics**:
   * Watch a beginner Flutter tutorial (e.g., “Flutter Crash Course” on YouTube).
   * Focus on Dart basics (variables, lists, functions) and Flutter widgets.
3. **Start Small**:
   * Build the Summary Page first (a list of projects).
   * Add SQLite later for data storage.