ChatGPT

ParrisDubboMover (PDM) Design Blueprint

A. Context & Constraints Summary

Josh Parris and his wife Kristy are relocating their young family (Sylvie, 3, and Elias, 1.5) from Bendigo, VIC to Dubbo, NSW in January 2026 for Josh's new job at Dubbo Christian School (DCS). This is a complex interstate move (~750 km) under tight timing: Josh's current role at La Trobe University ends by 24 Dec 2025, and DCS orientation starts by 20 Jan 2026 [1] [2]. The family must be living in Dubbo and settled by mid-January. Key constraints shaping the plan include:

- **Health & Family:** Kristy has multiple sclerosis (on Tysabri), so minimizing stress, fatigue, and disruption is critical [3]. Any lapse in her treatment or extreme exhaustion could trigger an MS relapse, so continuity of care and a calm moving process are paramount. The children are toddlers with special needs (Sylvie likely neurodivergent, on NDIS early intervention) [4], so maintaining their routines (sleep, childcare, therapy) and ensuring early education continuity for Sylvie is a high priority [5]. The move must accommodate medical handovers (neurologist, GP, NDIS providers) and new childcare arrangements without gaps.

- **Timing & Work:** The window for the move is narrow. Josh will work up to Christmas in Bendigo, then start at DCS in late January [1]. They set **15 Jan 2026** as the "drop-dead" date to vacate their Bendigo home (53 Buckland St) and hand it over to new tenants [6] [7]. Any schedule slip could jeopardize Josh's timely start at DCS. This necessitates precise planning of packing, travel, and temporary accommodations around fixed dates (e.g. truck rental, key handover, first work day).

- **Housing & Finance:** The family owns the Bendigo house and will convert it to a rental as they move out, and simultaneously must secure a rental home in Dubbo (targeting a 4-bedroom ~$580/week) [8]. They want to minimize double housing costs – any overlap of paying mortgage + Dubbo rent should be as short as possible [9] – yet some overlap is needed to allow a smoother move. Josh's salary (~$1,300/week net) means resigning even two weeks early would cost ~$2,600 in lost income [10], so they opted to **stay employed through late Dec** and avoid forfeiting pay. The move budget is tight; a full professional mover quote came in around $7–10k [11], so they are combining DIY efforts with selective professional help to keep costs in check [12] [13]. They have to allocate funds for fuel, a rental moving truck, possible storage, and overlapping rent, aiming to hold total moving expenses around $5–6k [14] [15].

- **Move Strategy:** After comparing four strategies (Options A–D) [16] [17], the family chose a **hybrid move plan ("Option C")** as the best balance of safety, cost, and low stress [18] [19]. In this plan, Josh works in Bendigo until Christmas, they begin a Dubbo lease in late Dec/early Jan (a short overlap of ~2 weeks), and execute the move using both a hired moving truck (for large items) and DIY trips for smaller items [20] [21]. This grants a small buffer if things go wrong while avoiding both the high expense of all-professional move and the unsustainable strain of an all-DIY move [22] [23]. The truck is booked from 13–23 Jan, with family help enlisted as drivers [24]. Key move milestones are already locked-in: truck pickup on 13 Jan, loading on 14 Jan, Josh and his dad (Michael) driving the truck to

Dubbo on 15 Jan, and Josh starting work on Monday 19 Jan [25]. These fixed points create an immovable backbone for the plan.

- **Logistics & Support:** The move involves co-opting family and friends across two states. For example, Josh's parents and in-laws (Michael, Rick, Sandy) are helping with packing, driving, and childcare around the move week [26] [27]. Overlapping commitments (a family beach trip to Jan Juc, a theatre event on 11 Jan for Kristy) complicate who is available when [28] [29]. The planning must coordinate multiple vehicles (a rental truck, the family's SUV and car, possibly a trailer) and multiple travel waves. There are open questions about driver assignments, passenger arrangements (each kid needs an adult in their car), overnight stops during the long drive, and handling of pets (e.g. four backyard chickens) [30] [31]. These uncertainties need resolution to avoid last-minute chaos.

- **Scope of Tasks:** The family's "to-do" list spans *at least 11 domains of life*. Beyond the physical move, they must manage: renting out their Bendigo house, finding a Dubbo home, switching utilities and internet, updating addresses with banks/government, transferring Kristy's job or finding a new one in Dubbo, enrolling kids in preschool/daycare, migrating health services (MS infusions, GPs, specialists), continuing Sylvie's therapy schedule under NDIS, obtaining new licenses/registrations in NSW, connecting with a new church and community, and more [32] [33]. They have assembled numerous documents and checklists to track these, from an 8-10 week pre-move countdown to an itemized inventory of things to pack and admin to finish [34] [35]. Managing so many threads is overwhelming (Josh has ADHD and easily feels the chaos of "too many loose threads" [36]), so a central "mission control" app is needed to bring order to the planning.

Given this context, the ParrisDubboMover (PDM) app will serve as a single, integrated platform to coordinate all aspects of the move. It must operate on both mobile (for on-the-go updates) and desktop, support offline or low-connectivity use (e.g. during travel), and be simple enough for a sleep-deprived couple to use. All planning information — currently scattered across Google Docs, PDFs, and email threads — should be uploaded into PDM so it can generate a personalized timeline, checklists, and decision-guiding insights tailored to the Parris family's unique needs. Constraints such as health (no gap in MS treatments), child wellbeing, hard deadlines, and budget limits will be embedded into the app's logic so that **"What do we do next?"** never has to be answered from scratch. In short, PDM is designed as the family's always-accessible project manager for the move, enforcing their constraints and priorities at every step.

B. Domain Model & Module Map

PDM is organized into modules corresponding to each major domain of the move, each aggregating relevant data, tasks, and decision support from the uploaded documents. The domain model centers on a few core entity types – **Documents**, **Tasks/Checklist Items**, and specialized entities like **Properties**, **Options**, or **Providers** – which together map to the life domains the family must manage. Below is a breakdown of each module's purpose, the core entities and user stories it involves, and how it draws on the source documents:

1. Housing & Rentals

Purpose: Help the family secure a new rental in Dubbo and prepare their Bendigo house for renting out. This module tracks property options and related tasks.

Core Entities: `RentalProperty` (for each candidate Dubbo house), `Task` (for Bendigo house prep and rental logistics). Key attributes for a RentalProperty include address, rent, size, status (e.g. "inspection scheduled", "application submitted"), and notes on pros/cons (distance to DCS or church, etc.). Bendigo home tasks (cleaning, repairs, agent handover) are represented as tasks linked to the Bendigo property.

User Stories: As Josh, I can input and compare shortlisted Dubbo rentals (4-bedroom homes around $500–$600/wk) [37], recording details like rent, inspections dates, commute distance, and suitability for family/church needs. I can see at a glance which rentals I've applied for and their status. I can also track tasks for getting our Bendigo house (53 Buckland St) ready for tenants: e.g. schedule cleaning, complete repairs, arrange photography, and engage the property manager.

Doc Linkage: This module synthesizes info from the *"Moving Options A–D"* doc about housing timing and costs, and the *"Move Checklist"*. For example, PDM will remind the user of the **rental overlap constraint** – that they only want to pay double housing for ~2 weeks [38] – when scheduling lease dates. It preloads tasks like *"Rental appraisal with agents"* (noting that Yolena from Hutton and Kate from Ray White inspected on 7 Nov [39]) and *"Clean and photograph Bendigo house before 15 Jan"*. From Option C's plan, it knows the **deadline for vacating Bendigo is ~15 Jan 2026** [6], so all prep tasks for 53 Buckland St must be completed by that date. The module also integrates insights from the option comparison – e.g. reminding that a longer overlap, while convenient, would cost an extra ~$580/wk in rent [40]. This helps Josh and Kristy weigh trade-offs as they consider lease start dates.

2. Kristy's Work & Career in Dubbo

Purpose: Compare and track Kristy's employment options in Dubbo, aligning them with family life. Kristy, a nurse, is evaluating job opportunities such as joining the Tresillian early parenting center, the local health service's casual pool, or private clinics.

Core Entities: `JobOption` (representing each job opportunity for Kristy) and associated decision criteria (pay rate, hours, flexibility). Also `Task` for any actions (applications, follow-ups) related to her job.

User Stories: As Kristy, I want to compare potential jobs side by side – for example, *Option A: Tresillian Dubbo, 2 days/week (8:30am–5pm)* vs *Option B: NSW Health Hospital casual shifts* vs *Option C: Private practice clinic* – so I can choose a role that fits our schedule and income needs. I can view how each job's schedule would sync with Elias's daycare days and Sylvie's preschool. I should be able to track applications or prerequisite steps (e.g. transferring nursing registration to NSW, contacting recruiters).

Doc Linkage: The app leverages hints from our docs about Kristy's likely work pattern. For instance, an email in the daycare file notes Kristy will "probably be working 8:30am–5:00pm at Tresillian in Dubbo" two days per week [41]. PDM uses that as one scenario, highlighting that if she works at Tresillian on Mondays and Tuesdays, those become fixed daycare days for Elias [42]. Another option could be a NSW Health casual pool with variable shifts – the app would warn that irregular hours might conflict with childcare availability (a fatigue risk for Kristy). The *module draws on the "Daycare Options" doc and schedule* to show, for each job option, whether the needed childcare coverage is feasible. In short, it cross-references **Kristy's work days vs. childcare slots vs. Josh's off-day** to ensure the chosen job is workable [43]. If Kristy has expressed preferences (e.g. wanting a job that doesn't exceed 2 days/week to manage MS fatigue), the app flags any option that violates that (like a full-time role) as likely unsuitable. While our documents don't list all possible

jobs, PDM's structure allows adding entries (with fields for shift times, commute, pay, etc.) and will incorporate Kristy's notes or advice (such as pay comparisons or stress levels) once provided.

3. Childcare & Schooling (Sylvie & Elias)

Purpose: Coordinate the children's daycare and preschool arrangements in Dubbo, ensuring both kids are cared for on the days Josh and Kristy work. This module handles finding a daycare for Elias (under 3) and preschool for Sylvie, and aligns schedules with the family's work roster.

Core Entities: `ChildcareOption` (for long-daycare centers that take under-3s) and `PreschoolOption` (for Sylvie's preschool/kindy). Each has attributes like age range, days available, cost, and status (enrolled, waitlisted). Also `Task` items for actions like enrollment paperwork, waitlist follow-ups, subsidy updates.

User Stories: As parents, we want to identify the best daycare for Elias for Mondays and Tuesdays, and secure Sylvie's preschool placement, ideally at DCS. We need to track which centers we've contacted, fees, and whether they can accommodate our "3 days per fortnight" pattern. We also want a calendar view showing who is where each day of the week (e.g. Week A vs Week B) so we never double-book ourselves or leave a gap in kid coverage.

Doc Linkage: This module heavily uses the *"Elias Mon/Tues Daycare Options"* doc. It already lists strong candidates: **Imagine Childcare Blueridge** (near DCS) [44], **Community Kids Dubbo** (which could take both Elias and Sylvie together) [45], **Goodstart Early Learning** (several centers) [46], and council-run options [47]. PDM presents these options with their pros/cons as outlined (e.g. Community Kids "could take Elias now *and* Sylvie, so both kids are in one place" [48]). For each option, the app may show a map or distance from key locations (home, DCS, Tresillian) and whether it fits the needed days. The module incorporates the unique **Week A/Week B schedule**: from the docs we know Josh has every second Tuesday off and Kristy plans to work fixed days (likely Mon/Tue) [42] [49]. The app therefore models a fortnightly cycle: on "Week B" Tuesdays, it marks that **no daycare is needed for Elias** (since Josh is home) and optionally Sylvie could also stay home or still attend preschool for routine [50]. This avoids paying for care unnecessarily on those days. The app will track Sylvie's **Dubbo Christian Preschool** application status and a backup plan if she doesn't get in (perhaps enrolling her temporarily in the same center as Elias) [51]. It can remind the user to confirm Sylvie's enrolment and update the Child Care Subsidy details with Centrelink [33] [52]. Example tasks auto-loaded from docs include: *"Email Dubbo Christian Preschool to clarify from what year Elias can enroll (born May 2024)"* [53], *"Put Elias on waitlists at chosen daycare(s) by Nov"* [54], and *"Decide whether to keep both kids in one center initially or split between preschool and daycare"* [51]. The UI might offer a simple timetable view: e.g., **Mon & Tue** marked as "Kristy work, Elias at daycare, Sylvie at preschool" and highlight that on alternate Tuesdays Josh is off (so he might pick up Sylvie early or keep Elias home).

4. Health & MS / Neurology Transition

Purpose: Ensure a seamless handover of Kristy's MS care and the family's healthcare needs to Dubbo providers. This module tracks the critical medical tasks: finding a new neurologist, transferring infusion treatments, getting referrals, and setting up regular healthcare (GP, pediatrician, etc.) in Dubbo.

Core Entities: `Provider` (for health providers like neurologist, GP, pediatrician, therapist), and `Appointment` (for any scheduled medical appointments or infusion dates), plus related `Task` (for referrals, obtaining records, etc.).

User Stories: As Kristy (or Josh managing her care logistics), I need to secure a neurologist in the Dubbo/Western NSW area who can continue my Tysabri infusions *without interruption*. I want to see a list of potential neurologists or clinics, with notes on their locations and specialties, so I can decide who to contact. I also need checklists for transferring all our medical records and setting up initial appointments (e.g. ensure the kids have a new GP and we have scripts moved to a Dubbo pharmacy).

Doc Linkage: PDM is informed by the *"Neurologists Dubbo for Kristy MS"* research doc. It will present options like the **Dubbo Health Service Neurology Clinic** (public clinic in Dubbo) and specialists who serve the area: e.g. *Dr. Ruhaida Daud* (neurologist visiting Dubbo) [55], *Central West Neurology (Orange)* which has MS specialists [56], or *Dr. Kieren Po* (Tamworth-based, visits Dubbo, MS focus) [57]. Each entry can show notes (e.g. "visits Dubbo monthly, special interest in MS, contact info") as listed in the document. The module will checklist all steps needed to transfer Kristy's treatment: for example, *"Obtain referral from current neurologist (Dr. Atvars) to Western NSW specialist"*, *"Ensure Bendigo infusion records and latest MRI/JCV results are sent to new clinic"*, and *"Confirm Western Cancer Centre Dubbo can administer Tysabri on schedule"*. Indeed, the docs specify **Dr. Rosamind Scobie** (Kristy's current doctor) needs to initiate an eReferral to Dubbo Base Hospital's oncology unit for Tysabri [58]. PDM highlights this as a **must-do task** and can store the contact info (Dubbo Base infusion unit at 170 Myall St, ph. (02) 6809 6200 [55] [59]). It will also log the first Dubbo infusion appointment once booked – e.g., if the plan is to get a January infusion in Dubbo, that date goes into an Appointment entry so it's visible on the calendar. The module covers other health admin from the checklist: transferring GP files, finding a new dentist and pharmacy, and updating Medicare/MyGov addresses [32]. It will flag **"no gaps"** for critical items (e.g. *"Ensure next Tysabri dose due late Dec/early Jan is scheduled in Dubbo or Orange – do not skip!"*). Essentially, this module serves as a medical move coordinator, preventing any lapse in care or missing paperwork during the move.

5. Josh's Work at DCS (Onboarding & DCS Companion)

Purpose: Help Josh transition into his new job at Dubbo Christian School smoothly. It provides an onboarding checklist for his new ICT/Library role and also acts as a mini "DCS Companion" with key info about the school (drawn from the deep-dive research report) and quick links/resources he'll need as staff.

Core Entities: `Task` (for pre-start onboarding steps and post-start to-dos), `Contact` (for important people at DCS), and possibly `Note/Resource` for storing reference info (like school policies or frameworks).

User Stories: As Josh, I want to make sure I've completed all HR and IT onboarding tasks before my first day at DCS. I also want a handy directory of key staff contacts and notes on DCS's systems and culture, so I feel prepared. For example, I should see tasks like "Submit signed contract and paperwork to DCS HR", "Complete mandatory child safety training module", "Set up staff email and IT accounts", etc. Once I start, I'd love a section in the app where I can quickly find DCS information (like the staff directory, term dates, or the educational creed) without digging through emails.

Doc Linkage: The *"Move Checklist"* and our notes highlight some pre-start tasks. For instance, about a month out, Josh needs to **notify La Trobe and return equipment** and **confirm onboarding with the new employer** [60] [61] (the docs mention BCCS induction, but in reality this translates to DCS onboarding). PDM will list: *"Complete exit formalities at La Trobe (return laptop, submit HR forms) by early Jan"* and *"Attend DCS HR induction call prior to Jan 19"*. On the DCS side, the module is enriched by the *"DCS Deep Research Expansion Report (2025)"*. From that, the app can include a **"DCS Companion"** section with bullet-point notes on

governance, key staff, and systems. For example, it might list the **Principal (Paul Arundell, retiring end of 2025)** and incoming principal, the **Business Manager (Scott Morris)** [62], the ICT Manager and other colleagues Josh might work with [63]. It can also include DCS's core values or frameworks like the *Transformation by Design* curriculum model [64] so Josh can review the ethos he'll be supporting. Links to important resources (e.g. staff portal, library system, or Berakah policies) can be stored for quick access. Essentially, this module blends task management (the things Josh must do before and just after Day 1) with an informational hub (adapted from the research doc and potentially from the existing DCSCompanion prototype he built). For instance, if the prototype had features like a staff directory or professional development (PD) tracker, those will be incorporated so Josh can continue using them in PDM. The goal is that on Day 1, Josh can open this app and have at his fingertips: a checklist of onboarding tasks (with due dates like "submit WWCC by first week" – see Module 7 below), and a cheat-sheet of who's who and how the school operates, distilled from the deep-dive report.

6. NDIS & Therapies for Sylvie

Purpose: Manage the transition of Sylvie's therapy services and supports under the National Disability Insurance Scheme (NDIS) from Bendigo to Dubbo. Sylvie currently receives services (occupational therapy, speech therapy, psychology or play therapy) that need to be handed over or re-established in Dubbo.

Core Entities: `Provider` (reused for therapy providers – e.g. new OT, speech therapist in Dubbo), and `Task` (for coordinating reports, referrals, and intake with those providers). Possibly an `Appointment` for any scheduled consultations.

User Stories: As parents, we need to ensure Sylvie's support continues seamlessly. I want to see a list of her current providers in Bendigo and note which reports or discharge summaries we need from each. I also want to research and track potential new providers in Dubbo (or nearby) for each therapy, including any waitlist or initial consult dates. The app should remind us of any scheduled calls (for example, we already have a planning session booked in Dubbo) and keep her NDIS information up-to-date (like notifying NDIS of address change).

Doc Linkage: The docs provide some specific guidance. The *"Move Checklist"* says *"Ensure NDIS providers for Sylvie are updated or re-established in Dubbo"* [32] – so PDM will include tasks like *"Update address and new circumstances in NDIS portal"* and *"Inform Sylvie's Bendigo therapists of move, obtain written progress reports."* We know from our notes that a **parent planning session with an early intervention specialist (Renee Matheson, Psychological Solutions Dubbo) is booked for 12 Dec at 4pm** [65]. The app will surface that as an upcoming appointment (and perhaps even show it on a calendar view in December). It will list out target providers in Dubbo for each therapy: e.g., if there's a known pediatric OT practice or speech pathology clinic in Dubbo, those can be listed as options to contact (the content may come from prior research or can be added by the user). Additionally, general tasks like *"Get copies of Sylvie's latest OT and speech reports for new providers"* and *"Ask NDIS planner about transferring funding to new region"* can be pre-loaded from standard moving checklists. This module essentially keeps **Sylvie's developmental support on track**, preventing any lapse. It might also integrate with the Health module for overlapping items (for example, Sylvie's pediatrician could be tracked as a Provider here or in Health). Given that continuity of care is a family priority, the app could flag this module's tasks as high priority – e.g. *"NDIS Early Intervention: High – ensure new therapy appointments are arranged within 1 month of moving"*. The timeline from the docs (6–8 weeks before move) included updating NDIS and researching Dubbo therapists [66], so PDM sets those tasks due accordingly (around Nov). If Sylvie has any specific needs (like particular therapy goals), the user can note

them in this module to discuss with new providers. By collating all this, the family won't lose track of any therapy in the upheaval of moving.

7. Licensing, Rego, WWCC & Number Plates

Purpose: Guide the family through required government/licensing changes when moving interstate. This includes transferring driver's licenses and car registrations from VIC to NSW, obtaining new NSW number plates for their vehicles, and getting NSW Working With Children Checks (WWCC) as required for their jobs (both Josh and Kristy will need WWCC for school/nursing roles).

Core Entities: Mostly represented as `Task` sequences or perhaps a mini workflow for each process (license transfer, vehicle rego, WWCC). Could also have a `Document` reference for forms (like if they upload their filled PDF forms or confirmation letters).

User Stories: As Josh or Kristy, I want to know exactly how to change our licenses and car rego to NSW plates, and what steps and documents are required (IDs, fees, inspections). I also need to apply for NSW WWCC clearances promptly so that our employment is not held up. The app should walk me through each step (e.g. "Book appointment at Service NSW in Dubbo or Echuca for license transfer") and let me check them off as completed.

Doc Linkage: The plan notes highlight this as a dedicated category. PDM will provide a step-by-step checklist for each of these bureaucratic tasks [67] . For example, for **Driver's License Transfer**: *Step 1:* Get a NSW driver's license within 3 months of moving – requires proof of identity, fill out form, surrender VIC license, pay fee. *Step 2:* Update car registration – likely needs a Blue Slip (vehicle inspection in NSW) and proof of garaging address, then get NSW plates. The app can list what to bring (e.g. VIC registration papers, ID, proof of address) and track fees paid. Similarly, for the **Working With Children Check (WWCC)**: since Josh and Kristy are moving from VIC to NSW, they must apply for new WWCC clearances in NSW. The docs mention a planned *"Echuca trip"* for processing this [68] – implying they might drive to the NSW border (Echuca/Moama) to do the ID check before moving. PDM will include: *"Apply for NSW WWCC online (do ASAP, as approval can take weeks)"* and *"Visit Service NSW (e.g. Moama) for WWCC ID verification – booking on [date]"*. Once completed, it can prompt to save the WWCC number/expiry in the app for reference. Each subtask (license, rego, WWCC) will have a status toggle (e.g. "Pending", "Submitted", "Completed"). The module might also include *vehicle-related tasks* like updating roadside assistance or getting a NSW e-tag for tolls [69] , but the core focus is the mandatory government paperwork. By aggregating these, the app ensures the family doesn't become unlawfully unlicensed or miss a requirement (for instance, working at a school without a valid WWCC would be an issue, so the app flags it with high priority). It draws from any instructions we have (if any doc has specifics, otherwise it uses general knowledge) and allows entering details like new plate numbers or license numbers once obtained.

8. Utilities & Services Setup

Purpose: Handle the disconnection and reconnection of utilities and essential services in the move from Bendigo to Dubbo. This covers electricity, gas, water, internet, waste services, insurance, and updating various addresses (banks, government agencies, subscriptions).

Core Entities: `Task` for each service to cancel or start, possibly a `ServiceProvider` entity (for utility companies, ISPs, etc.) if we want to store contact info/account numbers. Also might involve `Document` uploads for confirmation receipts (like new insurance policy or mail redirection order).

User Stories: As the planner, I need to ensure we won't be paying for utilities after we leave Bendigo and that everything is up and running in Dubbo when we arrive. I want a checklist that covers notifying our utility companies of move-out/move-in dates, setting up mail redirection, changing our address with banks and agencies, and transferring or closing any local services. I'd like to see, for example, that our electricity in Bendigo is scheduled to shut off the day after we move, and that it's turned on in Dubbo before we get there.

Doc Linkage: The *"Move Checklist"* provides a comprehensive list in categories, which PDM uses to populate this master checklist [70] [71]. For instance, under **Housing/Property** it lists: *"Cancel or transfer utilities (electricity, gas, water, internet, bin collection)"* [70] – PDM turns each of those into an item where the user can input the provider and set a disconnection date. The app will have paired tasks for connecting the same services in Dubbo. It also reminds to *redirect mail via Australia Post* [72] – an item the user actually completed (per notes, mail redirection from 23 Jan 2026 for 12 months was set up at ~$200 [73]). The app can record that confirmation number or simply mark it done. Finance/admin updates are included: *"Update Bendigo Bank address"*, *"Update Centrelink, ATO, MyGov, Medicare, electoral roll"* [74] all appear in the checklist and will be mirrored in PDM. Essentially, the module breaks down into two phases: **Before the move** – schedule disconnections and new connections; **After the move** – confirm everything is working and address changes are done. Based on timeline notes, PDM may schedule reminders (e.g. *"Confirm disconnection dates by 8 Dec"* [75], *"Disconnect Bendigo utilities on 20 Jan"* [76], *"Arrange internet at new house by 21 Jan"* [77], etc.). Insurance tasks are included too: e.g. *"Update home insurance for new address (effective move-in date)"* [78]. By aggregating these tasks, this module reduces the risk of forgetting a service (like accidentally paying two internet bills or failing to insure the new home). PDM can also store any account numbers or reference info the user enters, to generate a handy "change of address" letter or just to have everything in one place. Given how administrative these tasks are, the app's value is in *sequencing* them logically (for example, prompt to compare new utility plans 6-8 weeks out [79], confirm cutoff dates 4 weeks out [80], etc.) so that nothing falls through the cracks during the busy move week.

9. Packing, Decluttering, Logistics & Travel

Purpose: Orchestrate the physical move logistics – from decluttering decisions and packing progress to travel plans and coordinating who and what goes in each vehicle. This is one of the most detailed modules, essentially the "moving day(s) operations" planner.

Core Entities: `Task` (for packing milestones, item dispositions, and travel to-dos), possibly a `InventoryItem` for special large items being moved or not (like spa, trampoline, chickens), and a structured representation of the travel plan (which could be a `Trip` or simply a matrix of people/vehicles by date).

User Stories: As the chief mover, I want to track our packing status room by room so I know we'll be ready by load day. I also need to document decisions on big items (what we're selling vs taking) so that the truck space is planned. I want a clear travel plan for each segment of the move: who is driving each vehicle, who's riding with whom (especially for the kids), what cargo goes in each, and where stopovers will be. Ideally, I can generate a one-page run sheet for the move week to share with family helpers.

Doc Linkage: This module is built directly from the detailed move plans and brainstorming in our docs. For **packing and decluttering**, PDM uses guidance from the *"moving thinking"* notes and timeline. For example, it will prompt early decisions on bulky items: *"Decide fate of spa (sell or move?)"* – our notes indicate they're leaning toward **selling the spa in Bendigo** [81] to save cost/effort. Similarly *"Dismantle trampoline by early Jan (if taking)"* [81] and *"Decide whether to keep the kids' cubby house or leave for renters"* are tasks drawn from the doc. The app could have a mini list of **"Not Coming to Dubbo"** items. From the Move Plan Q&A: the four chickens are explicitly undecided, with a plan forming to give two away to friends and take two to Dubbo [31]. PDM can list *"Rehome 2 chickens with Katie-Ann & Will"* and *"Prepare travel crate for 2 chickens to Dubbo"* as tasks, ensuring even the pets are accounted for. It also covers the fate of the **outdoor spa** and **cubby** [31] (likely tasks to sell on Gumtree or arrange pickup). These decisions feed into packing: fewer items to move if sold/donated early.

For **packing tracking**, the module can present a checklist by room (possibly
"Living Room", "Master BR", "Kitchen", etc.) with an approximate box count or percentage. The *"moving thinking"* doc suggests aiming to close out 1–2 rooms per week in the lead-up [82] and leaving only essentials for last [83] [84]. PDM might implement this by letting the user mark rooms as "packed" or not. It will also include special tasks like *"Prepare a 'First Night in Dubbo' box"* [85] and *"Establish a 'Do Not Pack' corner for essentials"* [86], which were emphasized in the planning notes. These tasks ensure survival items are handy despite the chaos.

For **logistics & travel**, PDM introduces a structured "people/vehicle matrix" view. Based on the suggestion in the docs [87], the app will have a table for each trip leg (likely Bendigo→Dubbo on 15 Jan, and a second wave on ~18 Jan) listing: Date, Origin→Destination, Vehicle, Driver, Passengers, Key Cargo, Notes. For example, it will pre-fill an entry for **15 Jan**: Vehicle = "Rental Truck 18ft", Driver = Rick (picking up truck) then Michael (to drive to Dubbo), Passengers = (none, or Josh in cab), Cargo = "All household goods", Notes = "Depart Bendigo morning, drive ~9h to Dubbo" [88]. Another entry: **15 Jan**: Vehicle = "Michael's car", Driver = Josh, Passengers = (n/a, just Josh following truck), Cargo = "Plants or fragile items", Note = "Used for return trip if needed". Then **18 Jan (example)**: Vehicle = "Toyota RAV4", Driver = Rick, Passengers = Kristy, Sylvie, Elias, Cargo = "Chooks in crate, kids gear", Notes = "Likely stop overnight at West Wyalong" [89]. The idea is to capture exactly "who is where, in which car, on which date" [90], solving the mental juggling. PDM populates this with known info (from the Updated Plan doc: e.g. it knows *Truck pickup 13 Jan 4pm* [91]*, Load on 14th, Josh+Michael drive 15th, second group travels after cleaning on 15–17th* [92]) and leaves placeholders for open questions (e.g. "Who drives the Mazda?" [93] which is still unresolved). The app can highlight unresolved assignments in red, prompting the user to fill them in once decided.

Additionally, a **timeline view** (calendar or Gantt-style) in this module shows all key dates from early prep to post-move. PDM aggregates tasks and milestones onto a single timeline from November 2025 through February 2026. It uses the explicit dated schedule from the *"Master Checklist"* section [94] [95]. For instance, it will mark **23 Oct 2025** as "Decided on mid-Jan move window" [96] (maybe as an FYI past milestone), **7 Nov** for "Rental appraisal done" [97], **24 Dec** "Last work day (La Trobe)" and Christmas, **13 Jan** "Truck pickup 4pm" [98], **14 Jan** "Loading day", **15 Jan** "Josh + Michael depart (Trip 1)" [25], **16–17 Jan** "Cleaning house, second group departs", **19 Jan** "Josh starts at DCS" [99], etc., up to **23 Jan** "Truck returned to Bendigo" [100]. By visualizing this, the user can see how everything fits together week by week. The timeline is populated from document data (for example, tasks numbered 1–32 with relative timing in the checklist [34] [35] and the specific calendar in the doc [95] [101]).

Finally, the module will allow **printing or sharing** of relevant plans. For example, the user could print the packing checklist for sticking on the fridge, or export the vehicle matrix and timeline to share with their dad and father-in-law so everyone is literally on the same page. The docs even suggested making a Google Sheet for tracking tasks [102]; PDM will eliminate that need by being the interactive checklist itself, but it will support outputting a simplified checklist or schedule if helpers (who might not use the app) need it.

10. Church & Community in Dubbo

Purpose: Help Josh and Kristy plug into the faith community in Dubbo by presenting information on local churches and Christian networks. Moving away from their Bendigo church means finding a new church home; this module collates options and tracks their exploration of them.

Core Entities: `Church` (with attributes like denomination, size, programs, known connections), and `Task/Note` for visit plans or follow-ups (e.g. "Attend service at X church on Feb 2"). Possibly a `Contact` for any people they meet or already know in those communities.

User Stories: As a family serious about their Christian community, we want to identify a shortlist of churches in Dubbo to try out, learn about their style and connections, and keep notes on what we find. We also want to see if there are existing contacts or overlapping networks (perhaps through DCS or friends-of-friends) that could introduce us. The app should help us compare and remember key details as we visit each place.

Doc Linkage: The *"Dubbo Protestant Network: Hubs and Bridges"* report offers a goldmine of info on the Christian landscape in Dubbo. PDM will surface the **top churches and Christian organizations** from that network analysis. For example, it can list **Generocity Church** – noting it's a large ACC (Pentecostal) church that hosts citywide women's conferences and cross-denominational events [103]. It can list **Holy Trinity Anglican** – a historic church that acts as a bridge between traditional and evangelical groups, hosting combined prayer meetings [104]. It will include **Dubbo Baptist** – noting that they partnered with DCS (used the school hall during building works and hosted the 2025 combined prayer gathering) [105]. And **Dubbo Presbyterian** – an active congregation with lots of inter-church ministry involvement (Scripture Union camps, etc.) [106]. Even DCS itself is mentioned as a network hub [107]; while not a church, it means through the school Josh may automatically connect with multiple church communities.

The app presents each church with a summary (derived from the report): denomination, approximate size or influence, key joint activities, etc. It can also note if they have known ties to things the family values (e.g. if a church runs a playgroup or if many DCS families attend). The user can mark ones of interest ("shortlist") and schedule visits (perhaps tasks like "Visit Holy Trinity 10am service on Feb 7" or "Attend Generocity's next women's event" for Kristy). As they attend, they can add a quick note ("Kids liked it", "Good youth program", etc.) in the app.

Additionally, since the family has **existing connections** (for instance, Josh's role at DCS or any contacts via Cornerstone/Narromine mentioned in docs [108]), the app can remind them to leverage those. For example, *"Email Michael & Margaret (relatives nearby) about local church recommendations"* or *"Ask DCS colleagues which churches have many school families."* There was mention of *Cornerstone Dubbo/Narromine* [108] – possibly an existing community – which PDM would list so they remember to reach out.

Overall, this module ensures the **social/spiritual relocation** isn't forgotten amidst the logistics. It's less about tasks with deadlines (aside from possibly "notify our Bendigo church of the move" which was on the list [109] – likely already done or will do before leaving) and more about providing curated knowledge and a way to track subjective decisions (where do we feel at home?). It closes the loop by helping them find a supportive community, which is one of their values for a smooth transition [110].

11. "What Should I Do Next?" Engine

Purpose: Across all the above modules, this feature synthesizes the data (tasks statuses, deadlines, and priorities) to suggest the next 1–3 high-impact actions the user should focus on at any given time. It acts as an intelligent coach, guiding Josh (who often feels overwhelmed) to just a few next steps, always with context for *why* those are important.

Core Entities: This is more of a service than a data entity, but it heavily queries `Task` (filtering by due dates, priority flags, dependencies) and possibly uses an `AdviceRule` or the text embeddings of documents to generate rationale.

User Stories: As a stressed user, I want the app to tell me, *"Given everything on your plate, do these two things today."* It should pick tasks that are urgent or unblocked and explain why they matter now. For example, if we're 8 weeks out from the move, it might say "**This week: 1) Book the removalist/truck, 2) Schedule Kristy's infusion transfer.**" If I ask "Why these?", it should cite that the removalist needs advance booking in peak season [111] and that the MS treatment transfer should start 6+ weeks ahead to avoid any gap [35].

Doc Linkage: The logic of this engine is driven by the timelines and constraints in our documents. It will consider task lead times and critical paths. For instance, documents show that **booking a moving truck by late Oct/early Nov** was recommended [112] [113] – so if in the app that task isn't done by then, it becomes a top priority "Do Next" item, possibly highlighted in red. Likewise, **transferring Kristy's Tysabri infusion by ~6 weeks out** (early Dec) is critical [35], so as that date approaches, "What's Next" will bubble that up if incomplete. The engine can also use dependency logic: e.g., it won't suggest "hand back Bendigo keys" until after "finish cleaning the house" is done, because obviously cleaning precedes key handover. If the user is at moving week, it might prioritize immediate tasks like "Load the truck tomorrow – don't forget to set aside the first-night box" (with rationale that once the truck is gone, you need those essentials accessible). The rationale for tasks will be **grounded in the family's specific context** – often by quoting or referencing the documents they provided. For example, it might say: *"Next, call Dubbo Christian Preschool to confirm Sylvie's start date – this is urgent because preschool spots fill fast, and continuity is vital for her development [5]."* Or *"Next, update your address with NDIS on the portal, since provider searches can take time [66]."*

In implementation, this could be rule-based (if task due date is within X days or marked high priority and not done, suggest it) combined with a knowledge base lookup to pull a justifying tip from the docs (for instance, linking the mail redirection suggestion with the note that Josh already set it up for 12 months [114], acknowledging it's done). Potentially, an AI component (RAG – Retrieval Augmented Generation) could be used by feeding relevant document snippets to produce a friendly explanation for the suggestion. However, initially the engine can use simpler logic with static rationale text we derive from the docs.

The end result is that whenever Josh opens the app (or specifically the "What Next?" screen), he sees a gentle, authoritative prompt of what to focus on, *and he trusts it* because it's backed by their own planning

data and not generic advice. This feature will help cut through the noise of 100+ tasks to prevent paralysis by analysis, keeping the family on track day by day.

C. Data Model & Storage Design

PDM will use a lightweight relational database to store the moving plan data, ensuring consistency across modules. Below is a pseudo-ERD outlining the key entities and relationships (tables in a SQL sense) and how the uploaded document data ties in:

- **User** – Represents the user of the app (in this case, essentially Josh, or Josh and Kristy sharing one account). For the MVP we assume a single user, but we include this for completeness and future multi-user capability. Fields: `user_id`, name, email, etc. Each User has many Tasks, Documents, etc.

- **Document** – Each file the user uploads (PDF, DOCX, image) becomes a Document entry. Fields: `doc_id`, `title`, `content_text` (full extracted text), maybe `metadata` (upload date, file name). We will store the text so the app can search and reference it. (Optionally, an `embedding` vector could be stored for future AI semantic searches). **Relationships:** A Document *may spawn many Tasks* (if tasks are parsed from it or linked to it). For example, the Move Checklist doc would result in dozens of Task entries. We capture an association by either a foreign key in Task (task.origin_doc_id) or a join table between Document and Task, allowing tasks to link back to the source note for context.

- **Task** – The central table for actionable items and checklist entries. Fields: `task_id`, `description` (e.g. "Book removalist"), `domain` (category linking to module), `due_date` (if time-bound), `status` (pending/in-progress/done), `priority` (high, normal, low), `related_entity_type` & `related_entity_id` (optional linkage to a specific entity like a Property, Option, Provider, etc. that the task is about). **Relationships:** Task belongs to a User; optionally, Task may reference:

- a Document (as mentioned, to cite origin or attach reference text),
- a specific domain object (e.g. a task "Clean Bendigo house" might link to the Bendigo Property entity).

Many tasks will simply belong to a domain without a further entity. Domains can be indicated by an enum or a separate table:

- **Domain (Category)** – Could be a lookup table or just an attribute on Task to group tasks by module (Housing, Health, etc.). For flexibility, a `Domain` table with `domain_id` and `name` (“Housing & Rentals”, “Health & MS”, etc.) is useful. **Relationships:** One Domain to many Tasks. This makes filtering and displaying tasks per module easy.

- **Property** – Represents a house property (e.g. Dubbo rental options and the Bendigo house). Fields: `property_id`, address, type (e.g. "Dubbo rental candidate" vs "Bendigo primary house"), rent, notes, status. For Dubbo rentals, status might be “interested”, “applied”, “secured”. For the Bendigo house, it can store info like property manager contact or rental listing status. **Relationships:** A Property can have many related Tasks (via either a linking table or by tasks carrying a foreign key).

For instance, tasks “Repair drywall at 53 Buckland” or “Yolena to take photos” link to the Bendigo Property. Similarly “Inspect 12 Church St Dubbo” links to that rental Property.

- **JobOption** – Represents one of Kristy’s job possibilities. Fields: `job_id`, employer (e.g. "Tresillian Dubbo"), role, hours or FTE, pay_rate, pros, cons, status (e.g. applied, offered, decided). **Relationships:** Could link to tasks (like “Send application to Tresillian” or “Inform Tresillian of start date”), though these might simply be tasks in the Work domain referencing the job option. Possibly one JobOption can be marked as chosen. If Kristy decides on one, we might update its status and that could trigger tasks (like onboarding tasks for that job).

- **ChildcareOption** – Represents a daycare or preschool option. Fields: `childcare_id`, name (e.g. "Imagine Childcare Blueridge"), type (long-daycare vs preschool), min_age, max_age (for eligibility), location info, fees, and a field for which child (Elias or Sylvie) it’s for (or we may separate PreschoolOption vs DaycareOption classes internally). **Relationships:** Could have tasks (like “Tour facility” or “Join waitlist”) referencing it. Also might connect to a schedule or calendar entry (e.g. if chosen, store which days per week reserved). But initially we can keep schedule in notes and static pattern logic.

- **Provider** – A generic entity for service providers in Health/NDIS domain (and possibly could include utility companies or others, but let’s keep it to people/services). Fields: `provider_id`, name, type (e.g. "Neurologist", "GP", "OT", "UtilityCompany", etc.), contact info (phone, address). This can store doctors (Dr. Daud, Dr. Po, etc.), NDIS therapists, or even Yolena the property manager (type “Realtor”). **Relationships:** Providers can link to tasks or appointments. For example, a Provider entry "Dubbo Health Service Neurology Clinic" might be linked to an Appointment for Kristy’s infusion. A Provider "Western Cancer Centre Dubbo" might be linked to a task “Confirm transfer to WCCD”. By having providers in a table, the app can list them out (like a mini address book for the move). Many of these will be sourced from docs (the Neurologist doc provides several names to pre-populate).

- **Appointment** – Represents a calendar event (distinct from tasks because tasks can be done any time before a due date, whereas appointments happen at a specific time). Fields: `appt_id`, datetime, description, related_provider (optional), location. **Relationships:** Could link to a Provider (e.g. appointment with Renee Matheson on Dec 12) and/or to a Task (e.g. a task "Attend appointment" that can be checked off). But we might simply treat appointments as non-checklist calendar entries. The app will display them in timeline and can notify the user as they approach. Examples: the Dec 12 session with the psychologist, Jan 19 first day of work (not exactly an appointment but a fixed date event), or the truck pickup slot on Jan 13.

- **VehicleTrip / TravelPlan** – (Optional) A structured representation of the multi-vehicle travel matrix. We could create a `Trip` entity with fields: date, from_location, to_location. And a related child entity `TripAssignment` with vehicle, driver, passengers, notes linked to a Trip. However, it might be overkill to fully normalize this; we could store it as a specialized data structure or even just use tasks/notes for each leg. If time permits, a small table makes sense:

- `Trip`: `trip_id`, date, origin, destination.
- `TripAssignment`: `assign_id`, trip_id, vehicle, driver, passengers, cargo_notes, other_notes. This way the UI can reconstruct the matrix. This data might not come from uploaded docs directly (except

via manual entry of the decisions), so initially it could be user-entered. We design for it but it can be populated manually.

- **Decision / OptionChoice** – (Optional high-level, not a table per se) The fact that Option C was chosen could be stored as a config setting (like `chosen_move_option = C`). This might slightly alter app behavior (for instance, if they had chosen Option B full removalist, some modules like Packing would be simpler; but since Option C is locked in, the app emphasizes those tasks accordingly). In our case, we can hardcode the plan since it's confirmed [18]. No separate table needed; it's just context.

The **relationships summary**: Most tables connect via tasks. The Task table is a nexus linking to Domain (for categorization) and optionally to others. For example: - A Task "Mail redirection" is Domain=Utilities, no specific entity link, due date some time in Dec, source=Move Checklist doc. - A Task "Call Dr. Ruhaida Daud's office" might have Domain=Health, Provider=Dr. Daud, source=Neurology doc. - A Task "Submit WWCC application" Domain=Admin, could have a link to a Document if the user uploaded the WWCC PDF guide, but not required. - A Task "Inspect 14 Smith St rental" Domain=Housing, links to Property=14 Smith St.

Documents link to tasks that were derived from them, enabling traceability (the user can click a task and see a snippet from the original document that prompted it, helping them recall context – e.g. a note from the checklist or advice text [34]). We will maintain the citation or reference text for tasks that come directly from docs, possibly storing a snippet or line reference in the Task metadata.

All entities include a `user_id` to associate them with the user's account (in a multi-user future, or if the app was adapted for others). Data will likely be stored in an SQLite database (file-based, easy to set up for a solo user) which is sufficient for the scale of this project (a few hundred tasks, maybe a dozen documents). SQLite also supports full-text search which could be handy for querying document text.

For binary files (the actual PDFs, images), we might store them on disk or in the DB as BLOBs. Extracted text definitely goes into the DB for quick access. We will also generate and store **summaries or structured data** from documents upon upload. For example, when the *Options A–D* doc is ingested, we might parse out the fact that Option C was recommended [18] and store that in a settings table or at least highlight it in the UI.

To illustrate, here's a simplified schema outline in pseudo-code form:

```
User(user_id PK, name, email, ...)

Document(doc_id PK, user_id FK, title, content_text, source_path, upload_date)

Task(task_id PK, user_id FK, description, domain_id FK, due_date, status,
priority,
     origin_doc_id FK NULL, related_property_id FK NULL, related_job_id FK
NULL,
     related_provider_id FK NULL, related_childcare_id FK NULL, notes)

Domain(domain_id PK, name)  -- e.g., 1=Housing, 2=Work, ... 11=Church,
12=NextActions (if needed)
```

```
Property(property_id PK, user_id FK, address, type, rent, status, notes)
JobOption(job_id PK, user_id FK, employer, role, hours, pay_rate, status, notes)
ChildcareOption(childcare_id PK, user_id FK, name, type, location, min_age,
max_age, fees, notes, status)
Provider(provider_id PK, user_id FK, name, type, contact_info, notes)
Appointment(appt_id PK, user_id FK, datetime, description, provider_id FK NULL,
location, notes)

Trip(trip_id PK, user_id FK, date, origin, destination, notes)
TripAssignment(assign_id PK, trip_id FK, vehicle, driver, passengers,
cargo_notes, misc_notes)
```

This schema is **relational but flexible**. Many fields like `notes` allow storing unstructured info that doesn't fit elsewhere (ensuring we can capture anything unique from the docs). By using separate tables for things like JobOption, ChildcareOption, Provider, we keep domain-specific data organized. This avoids one giant tasks list with everything hard-coded; instead, tasks link to these when needed (so the UI can, for instance, on the Health screen show a list of Providers with related tasks under each – like a sub-checklist for each doctor transfer).

Storage considerations: SQLite will handle our needs offline. We'll use migrations (via a tool like Knex or Prisma in a Node environment) to define these tables. As the user uploads documents, we'll parse them (using, say, Python `docx` or PDF text extractors) and populate the Document.content_text. Then either automatically or via prompting, we'll create initial Task entries from that text (e.g. by regex patterns or with AI assistance to parse checklists). We can tag those tasks with origin_doc_id and domain, but leave them unassigned to specific dates until the user reviews/schedules them (or we infer dates from phrases like "8 weeks before move" etc., as seen in the checklist doc).

By maintaining a clear separation between raw data (documents, extracted facts) and planning data (tasks, events, decisions), the app allows continuous integration of new info. For example, if a new PDF (say a moving company quote) is uploaded, it could be linked to a task "Review moving company quote" in the Housing domain. Or if the user updates the plan (decides on a daycare), that reflects by updating the chosen ChildcareOption status, possibly triggering related tasks (like enrollment forms).

In sum, the data model is designed to mirror the real-world entities the Parris family is dealing with (houses, jobs, daycare centers, people, and lots of to-dos) and to maintain traceability to the original documents that inform each decision or task.

D. App Flow & UX Outline

PDM will be a responsive web application with a user-friendly interface that tired parents can navigate intuitively. The UX is organized around the modules (domains) with a unifying dashboard. Below, we describe the main screens and walk through representative user flows for each module, as well as the "What's Next" guidance and notification logic.

Main Screens & Navigation:
- **Dashboard/Home:** The landing screen after login shows a quick overview: a progress summary for each domain (e.g. X tasks done of Y in Housing, or a status like "Lease secured" / "Pending" in that module), and the top "Next Actions" suggestions prominently at the top. It might also show a countdown (e.g. "8 weeks until Move Day") and any upcoming appointments or deadlines this week. The tone is encouraging and focused.

- **Documents & Insights:** A section where the user can upload new documents or view a list of already uploaded ones. After upload, the app shows key extracted info (like a list of detected dates, or headings found). There may not be a separate heavy UI here beyond an upload dialog and a simple file list, since the value is in how docs populate the modules.

- **Module Pages:** For each domain (Housing, Work, Childcare, etc.), there is a dedicated screen or tab. The user can navigate via a sidebar or menu listing all modules A through 11. Each module page contains the content specific to that domain: e.g. forms, tables, and checklists described below.

- **Timeline/Calendar:** A calendar view (perhaps accessible via the dashboard or a menu) that shows all tasks and appointments chronologically. This helps in overall planning and is particularly useful around the move week. It can toggle between an Agenda list vs. a week grid.

- **Settings/Profile:** Minimal settings (maybe just account or an option to backup data). Possibly where the user can toggle that Option C is the plan (though that's default) or set the move date if it shifts.

Now, **UX flows per module:**

Housing & Rentals – Sample Flow:
Scenario: Josh wants to update the housing status after applying for a Dubbo rental and also check what's left to do for the Bendigo house. 1. **Navigate to Housing:** From the dashboard, Josh taps "Housing & Rentals". The Housing page has two panels: **Dubbo Rental Search** on top and **Bendigo House Prep** below. 2. **View Rental Options:** In the top panel, he sees a list of rental properties he entered. Each shows an address, rent, and status. For example, "123 Church St – $550/wk – **Applied**". One might be marked **Approved** (if they get a house, he can mark it accordingly). There's an "Add Property" button. 3. **Add a Rental:** Josh clicks *Add Property*. A form appears where he inputs: Address, Rent, Notes, and a few checkboxes (e.g. "Inspected?", "Application submitted?"). He enters details for a new listing they found. He hits Save – it now appears in the list. 4. **Compare Options:** If he clicks a property in the list, a detail view or modal opens, showing all info (commute distance to DCS, pros/cons fields he can edit, maybe an embedded Google Map). He can see side-by-side or just toggle through them to compare. The app doesn't algorithmically pick one, but by having all info consistent, it aids their decision. There might be a "Chosen " toggle to mark one as the final rental once decided. 5. **Bendigo House Tasks:** Scrolling down, the Bendigo section lists tasks like a checklist: *"Fix porch light (due Dec 20)"*, *"Clean and photograph house (due Jan 10)"*, *"Arrange key handover with agent (due Jan 15)"*. Each task has a checkbox. Josh sees *"Rental appraisal – Yolena (Done)"* already checked off (it was done on 7 Nov [97]). Suppose today he finished the minor repairs, he checks "Fix porch light" off – the item strikes through or moves to a done list. The list might have a progress bar for fun (e.g. "3/5 house prep tasks done"). 6. **Automated Reminder:** One task "Advertise listing by Dec 15" is overdue (say today is Dec 18). It's highlighted in red. The system earlier might have sent a notification or shown on dashboard that this needed attention. Josh marks it done if he actually already handled it with the agent offline. 7. **Insights on Deadlines:** If Josh tries to change a due date that goes beyond Jan 15 for a

Bendigo task, the app might warn “Careful: You plan to leave Bendigo by Jan 15 [6], so tasks should be done before then.” This is an example of constraint logic in action. 8. **Outcome:** Josh leaves the Housing page with an updated view of which rental is in play and feeling on top of the home handover tasks. The system might subtly indicate the next big housing milestone (e.g. “ Tenancy to start Jan 1 – ensure utilities are arranged (see Utilities module)” as a nudge).

Kristy’s Work – Sample Flow:
Scenario: Kristy wants to log her decision to work at Tresillian two days a week and see what that entails. 1. **Navigate to Work module:** Kristy taps “Kristy’s Work & Career” in the menu. 2. **View Job Options:** She sees a comparison table or cards for each JobOption entered. For example: - **Option 1: Tresillian Dubbo** – Hours: 2 days (8:30–5), Pay: $X/hour, “Pros: familiar work (early parenting), likely low stress; Cons: only part-time, lower pay”. - **Option 2: NSW Health Casual** – Hours: up to 2 days, but variable shifts, Pay: $Y/hour (higher); Pros/Cons listed (higher pay but unpredictable schedule, could clash with childcare). - **Option 3: Private Clinic** – etc. Each card might have an icon or color indicating viability (the app could subtly mark Tresillian as green since it aligns with the Mon/Tue daycare pattern we set, whereas the casual pool might be yellow due to schedule uncertainty). 3. **Select Option:** Suppose Kristy has decided on Tresillian. She clicks a “Choose this option” button on that card. The card gets a badge “Chosen”. The module might then show an expanded checklist specific to that job: e.g. *“Complete HR onboarding for Tresillian”*, *“Provide vaccination proof to employer”*, etc. If she had chosen the hospital option instead, the checklist might differ (maybe “register with NSW Health system”). 4. **Check Schedule Alignment:** There’s a small schedule preview on this page: something like “Your chosen pattern: Mon & Tue work. Childcare coverage: Elias at Imagine Mon/Tue, Sylvie at DCS preschool Mon/Tue.” If something was not lined up (say she chose a job needing 3 days but daycare is only 2 days), the app would flag that: “⚠ Need to adjust childcare for Wed if taking this job.” 5. **Add Notes:** Kristy can tap on an option to add notes or update details. For instance, she updates Tresillian’s status to “Offer accepted” and notes her start date if known. 6. **Outcome:** Having finalized her choice, PDM may generate new tasks like *“Email Tresillian acceptance”* (if not already done), *“Add Tresillian induction date to calendar”*, and *“Coordinate with daycare to finalize Elias’s schedule”*. This module essentially now transitions to helping her onboard that job (some overlap with Utilities for address changes if needed, etc., but mainly contained here).

Childcare & Schooling – Sample Flow:
Scenario: The family got a spot at Imagine Childcare for Elias; they need to update that and figure out Sylvie’s preschool outcome. 1. **Navigate to Childcare module:** They open “Childcare & Schooling”. 2. **View Options List:** The UI shows a list or grid of childcare centers with key data. Possibly an interactive map with pins could be there (not essential, but a nice touch given addresses). Initially, entries like Imagine, Community Kids, Goodstart are listed with a short blurb from our doc (capacity, age range). 3. **Update Elias’s Enrollment:** Josh taps “Imagine Childcare Blueridge”. He sees details and an “Enrolled?” toggle. He switches it to Enrolled and inputs the start date (e.g. 20 Jan 2026, Mon/Tue each week). The app might prompt “Confirm pattern: Mon every week, Tue every week except alternate Tuesdays when Josh is off? [Yes/No]”. He confirms yes – which effectively marks in the app’s schedule that every second Tuesday is off (this could integrate with Calendar or just be logic). 4. **Sylvie’s Preschool:** There’s a section for Sylvie: perhaps a dropdown or radio to indicate “Sylvie’s 2026 plan: ( ) DCS Preschool, ( ) Other/Not decided”. If they applied to DCS Preschool, PDM might list “Dubbo Christian Preschool – application sent, outcome pending”. If by now they got confirmation, he can mark it “Accepted”. If not, there could be a note “If not accepted, consider backup: enroll Sylvie at [Community Kids] with Elias.” He could then decide Option A or B as mentioned in doc [51]. Say Sylvie did get a spot at DCS for Mon/Tue. He marks that, and the app then shows Sylvie: Mon/Tue at DCS Preschool starting late Jan. 5. **Schedule Table:** Given these inputs, the module might show a

simplified weekly schedule (perhaps Monday through Friday columns, highlighting who goes where on Mon/Tue). For example, Week A: **Mon** – Kristy work, Sylvie DCS, Elias Imagine; **Tue** – Kristy work, Sylvie DCS, Elias Imagine; **Wed/Thu/Fri** – normal (maybe Sylvie home or other activities). Week B: same except Tue – Josh off, so Elias home. This is displayed textually or as a little calendar. This reassures them the plan works out. 6. **Tasks & Next Steps:** The lower part of the screen lists remaining tasks: e.g. *"Complete enrollment forms for Imagine by Jan 5"*, *"Pay bond/deposit for childcare"*, *"Orientation visit at Imagine - schedule?"*, *"Buy Sylvie's preschool uniforms (if any)"*. If any are done, they check them off. One task might be *"Notify Centrelink of new childcare for CCS"* [33] . If they click that, details pop up about updating the subsidy and perhaps a link to the Centrelink portal. 7. **Outcome:** The family has a clear picture that childcare is sorted. Any unresolved item (like if Sylvie's spot were uncertain) stays highlighted. The app could even have a "Plan B" note visible (like "Both kids at Community Kids as backup if DCS Preschool waitlist doesn't clear by Jan 10"). This can be updated as needed.

Health & MS – Sample Flow:
Scenario: It's early December; Josh is using PDM to make sure Kristy's medical handover is on track. 1. **Navigate to Health module:** He opens "Health & MS Transition". 2. **View Provider List:** At the top, there might be a list "New Providers to Confirm" and "Current Providers to Wrap Up". Under "New Providers", it lists the candidates: - Dubbo Health Service Neurology Clinic – Status: Not contacted. - Dr. Ruhaida Daud – Status: Contacted 1 Dec. - Central West Neurology (Orange) – Status: On hold. - Dubbo Base Hospital infusion unit – Status: eReferral sent. (These could be line items with a contact icon to call or email, since phone numbers are stored [115] .) 3. **Mark Progress:** Josh clicks "Dubbo Health Service Neurology Clinic". He sees details (address 170 Myall St, etc.) and a sub-task list: *"Send referral here"*, *"Check if they have MS nurse"*, etc. If Dr. Scobie (in Bendigo) already sent the eReferral, Josh marks that task done. He maybe adds a note "Spoke to receptionist, they will call back with appointment date". 4. **Current Provider tasks:** There's an item "Dr. Atvars (Bendigo neurologist) – Provide final summary". Maybe Josh checks that as done once he's got the paperwork. 5. **Highlight Critical Task:** The app has highlighted *"Confirm next Tysabri infusion appointment"* in red because it's due soon. Josh clicks it and sees a note: "Kristy's next infusion should be by first week of Jan. Ensure Dubbo or Orange has this booked [116] ." He realizes they haven't got confirmation yet – so that's the next call to make. PDM might even suggest it in the Next Actions. 6. **General Health tasks:** Scrolling, there are tasks like *"Find new GP in Dubbo – Book for late Jan"*, *"Transfer kids' immunization records"*, *"Locate a local pharmacy for meds"*. Some have due dates around the move or just after. He checks off "Downloaded kids' immunisation history" (because the doc said to do that by 13 Jan [95] [117] , and suppose he did it). 7. **Outcome:** This flow ensures no health item is missed. If something is pending that is time-sensitive (e.g. no neurologist confirmed yet by the time they move), the app will keep it top-of-list. Josh is able to update statuses as calls and referrals happen, and the app provides peace of mind by showing green checkmarks next to the most critical health needs once done.

Josh's Work (DCS) – Sample Flow:
Scenario: It's a week before starting work; Josh checks PDM to see if he's missed any onboarding steps. 1. **Navigate to DCS Work module:** He opens "Josh's Work at DCS". 2. **Onboarding Checklist:** Right away a checklist is visible: - "Sign and return contract to DCS HR" – done (he ticked it last month). - "Apply for NSW WWCC" – in progress (linking to Module 7, maybe shows status). - "Complete online training modules" – not done (due by 18 Jan). - "Set up new email account" – pending (will do on Day 1 maybe). - "Bring ID docs on first day" – reminder. Each item might have an info icon; e.g. clicking WWCC shows "Applied on 15 Nov, awaiting clearance – **Remember to bring WWCC number on Day 1**". 3. **DCS Companion info:** Below tasks, there might be expandable sections: - " Quick Links": a list of hyperlinks (DCS staff portal, library system login, etc. pulled from the research doc or known resources). - " Key Contacts": a mini directory – e.g.

Principal: [Name, picture if available, note “new principal starts 2026”], ICT Manager: [Name], HR contact, etc. - “ Notes from Research”: e.g. a snippet: “DCS is governed by Berakah Christian Education Ltd, a parent-controlled board [118]. The school has ~700 K-12 students + 120 preschoolers [119]. Principal Paul Arundell retiring end 2025, new principal starts Jan 2026 [120].” This gives Josh conversational knowledge to ease him in. - Perhaps “🗂 Christian Education Frameworks”: a bullet about “Transformation by Design curriculum model” [121], “Statement of Faith” etc., as quick refreshers. 4. **Check WWCC Task via Another Module:** If Josh clicks the WWCC task, the app might navigate or pop up the detail from Licensing module showing its status (some cross-module linking to avoid duplication). 5. **Outcome:** Josh finds he’s on track. The only thing blinking might be “Complete training modules” – he clicks it, sees details (maybe the DCS HR had given him a link to some online induction). He realizes he should do it this week. The app doesn’t do it for him, but since it’s flagged, he schedules time.

NDIS & Therapies – Sample Flow:
Scenario: It’s after moving, in February. Kristy wants to check Sylvie’s therapy schedule and ensure new providers are in place. 1. **Navigate to NDIS module:** She opens “Sylvie’s NDIS & Therapies”. 2. **Provider list:** She sees entries like: - Bendigo OT – status: “Provided report, case closed.” - Dubbo OT – status: “Initial consult on Feb 10.” - Bendigo Speech – “Last session done, report received.” - Dubbo Speech – “Searching… (to find provider)” - Psychologist (Renee Matheson) – “Parent session done 12 Dec [65], follow-up scheduled 5 Feb.” Each of these might be represented similar to Providers in Health, but focused on Sylvie. 3. **Add Provider:** If they haven’t found a speech therapist yet, the app shows that as an open item. Kristy can click “Add provider” if she got a recommendation. She enters a new Provider for Speech Therapy Dubbo (with contact info). 4. **Appointments:** On a calendar snippet or list, it shows *“Feb 5: Renee (Psych) follow-up”* and *“Feb 10: OT Intake meeting”* etc. These were either imported from emails or manually added. 5. **Task list:** Includes *“Update NDIS plan address – Done”*, *“Enroll Sylvie in new OT – Done”*, *“Find new speech therapist – TODO”*. She clicks “Find new speech therapist” and maybe marks it in progress (the app could optionally integrate to search online but that’s beyond scope; it’s a manual note). 6. **Outcome:** Kristy confirms all critical therapy continuity tasks are done except speech, which is now top priority. The app’s next-action engine would certainly surface “Find a speech therapist for Sylvie” as a suggestion until they mark it done.

Licensing & Regos – Sample Flow:
Scenario: Right after moving (late Jan), Josh sits down with PDM to finish all the license/registration formalities. 1. **Navigate to Licensing module:** He opens “Licensing & Rego”. 2. **Step-by-step lists:** The UI might have an accordion or wizard-like list: - **Driver’s License to NSW:** Steps listed (with checkboxes): *“Fill out license transfer form”*, *“Get NSW license photo taken at Service NSW”*, *“Destroy old VIC license”*. Suppose he completed these in Dubbo on Jan 22, he checks them all. Each step might have a sub-note (like required documents). - **Car Registration to NSW:** Steps: *“Obtain Blue Slip (vehicle inspection)”*, *“Visit Service NSW to register car in NSW”*, *“Receive new NSW plates, install them”*, *“Cancel VIC rego (apply for refund)”*. He might have done the inspection but not the Service NSW visit yet. So one step is checked, the rest pending. The app highlights that the rego must be done within 3 months of residency (general rule) – maybe a note on top. - **WWCC Clearance:** Steps: *“Apply online”* (done), *“In-person ID check”* (done in Echuca in Dec), *“Receive WWCC number”* (pending). If he’s received an email with the clearance, he could enter the number in a text field here for record. Kristy would have a similar set (the app could either list both or just assume both do it; might be combined as one item “WWCC for Josh and Kristy”). 3. **Smart Contextual Help:** If a step has complex requirements, the app might have a “info” icon. Clicking it might show a tooltip like: “To transfer rego, you need a NSW address proof (lease or utility bill) [9], your VIC rego papers, and proof of CTP insurance.” We gather these requirements from known standards or could have been in a doc if included. If user uploaded a guideline PDF, even better. 4. **Notifications:** The app could schedule reminders. For

example, if they haven't completed rego by March, it might send an alert "Don't forget to register your vehicles in NSW before the deadline to avoid fines." 5. **Outcome:** Josh systematically goes through and marks things done as they complete them around the move date. The visual separation of each process ensures clarity (they won't confuse steps between license vs rego).

Utilities & Services – Sample Flow:
Scenario: Two weeks before moving, and then immediately after moving, PDM helps coordinate utilities. 1. **Pre-move (Dec):** Josh opens "Utilities & Services". He sees sections for each service: - **Electricity:** Provider in Bendigo: EnergyAustralia (example) – *"Schedule disconnect: Yes, on 20 Jan"* (set via date picker). Provider in Dubbo: Origin Energy – *"Connect on 10 Jan"* (if they arranged early, or maybe they'll do as soon as lease starts). If not set, a task "Arrange new electricity connection" is pending. - **Internet:** Bendigo: Aussie Broadband – *"Cancel on 20 Jan"*, Dubbo: *"Order NBN – pending"*. He clicks "Order NBN" task, enters that he called and setup install on 21 Jan. - **Water & Gas:** similar structure. - **Council/Garbage:** Perhaps a note "Dubbo City Council – new bins requested starting Jan 2026" or if renting, property manager handles it. - **Mail Redirection:** Shows " Mail redirect in place from 23 Jan 2026 to 23 Jan 2027 [73] ." - **Insurance:** "New home contents insurance – started 20 Jan, policy #… (entered)", "Update car insurance address – pending". - **Financial/Legal:** "Update bank address – pending", "Update Medicare – pending", etc. 2. **Update an item:** When Josh arranges internet, he marks that as done and inputs the new WiFi plan details in notes (optional). The task disappears from "pending" list. 3. **Post-move (Jan end):** After moving, Kristy opens this module to verify everything. Perhaps the app has a toggle "Show post-move checklist" which lists: *"Confirm all utilities active in Dubbo"*, *"Test internet"*, *"Return any rental equipment (modem) to old provider"*, *"Check final bills are paid"*, *"Hand back Bendigo house keys"* (that last one might appear here or in Housing). She checks off "Keys handed to property manager" which might also mark a task in Housing complete. 4. **Outcome:** All the admin heavy-lifting is tracked. When all items are checked, the module might display a friendly " All services transferred!" which is extremely satisfying after such a move.

Packing & Logistics – Sample Flow:
Scenario: It's the first week of January. Josh uses the app to plan the final week logistics and ensure packing is on target. 1. **Navigate to Packing module:** He opens "Packing & Logistics". 2. **Packing progress:** At the top, a progress bar or summary: e.g. "80% of house packed (estimated)" with subtext "Most rooms done, 2 rooms left". This might be computed from tasks or manually updated. Maybe each room is listed with a checkbox "Packed". As he packs the kids' room, he checks it off. 3. **Decluttering decisions:** A section lists "Decisions on bulky items". It shows Spa – *Decided: Sell (buyer picking up Jan 8)*, Chickens – *Decided: 2 stay, 2 go (arrange Jan 14)*, Cubby house – *Decided: Leave for renters*. If something was undecided, it'd show a red "Pending decision". He has now decided about the trampoline (take it, and he put "Disassemble by Jan 10" as a task). 4. **Vehicle matrix view:** The UI likely has a special "Move Plan" tab within this module that brings up a table. It lists Jan 13, 14, 15, 16, etc. For Jan 15, he sees two entries as described earlier (Truck, Michael's car). For Jan 18 (say the second trip) he sees RAV4 and Mazda entries. One entry might still have "Driver: ___?" blank for the Mazda on Jan 15. He taps it and selects himself as the driver (or maybe Dan, if that's possible). This fills the blank. Now all cars have drivers. The app might automatically adjust passengers (if Josh is in the truck then he's not driving the Mazda, etc., it ensures consistency). He also adds a note for Jan 15 truck: "Include esky for road – it'll be hot". 5. **Timeline/review:** There is a timeline view that can be toggled to from this module or from a global view. He opens the timeline to January. It shows entries on each date: - Jan 13: "4pm – Pick up 4-ton truck (Rick)". - Jan 14: "All day – Load truck (everyone)". - Jan 15: "7am – Josh & Michael depart with truck + car". And separate "15 Jan – House cleaning begins (Kristy, Sandy)". - Jan 16: "Continue cleaning (if needed)". - Jan 17: likely "Second group departs Bendigo". - Jan 18: "Everyone arrives Dubbo". - Jan 19: "7:50am – Josh's first day at DCS". The app might allow editing or adding

events here as needed. It pulled many of these from the plan docs. 6. **Printing for helpers:** Josh clicks "Export Plan" and selects "Travel assignments". The app generates a PDF or print view of the matrix and timeline for Jan 13–19. He can hand this to his dad and in-laws so everyone knows the plan (no one has to rely on memory or group texts). 7. **Outcome:** With this module, Josh can visually verify that the logistics puzzle is solved. All tasks leading up (packing, cleaning) are either done or scheduled. The family and helpers know who's doing what. This dramatically reduces last-minute confusion, and if any assumptions changed (say a helper gets sick), he can quickly adjust the matrix and timeline, and see the ripple effect (the app might flag if a driver is removed and not replaced, etc., keeping consistency).

Church & Community – Sample Flow:
Scenario: In February, after settling in Dubbo, the family uses PDM to keep track of which churches they've visited. 1. **Navigate to Church module:** Kristy opens "Church & Community". 2. **Church list:** She sees cards or a list of the top 5 churches from the research (Generocity, Holy Trinity, Dubbo Baptist, Presbyterian, OneLife, etc.). Each card shows maybe denomination and a "network" score or summary from the research doc. 3. **Log Visits:** She clicks "Dubbo Baptist". On the details, there's a place to add a note or mark as visited. She selects "Visited on Jan 28" and writes "Welcoming people, kids program was great. Might return." The card now highlights that as visited (maybe with a check or by moving it to top of list). 4. **Shortlist:** She and Josh had discussed Holy Trinity too; she marks that with a star "Want to visit". The app can sort or filter by shortlisted. 5. **Community connections:** There might be a section listing organizations or events. E.g. "Dubbo Christian Ministers Association – Citywide prayer nights" or "Woven Women's Conference – annual, hosted by Generocity [103] " as FYI. Or simpler: maybe just listing that DCS itself will connect them to many communities [107] . 6. **Outcome:** Over a few weeks, they visit the starred churches and record notes. Eventually they decide on one (say they settle in Dubbo Presbyterian). They mark it "Chosen church" (if we allow that flag). The app doesn't do much with that except perhaps congratulate them and maybe in future could integrate schedules for that community (not in scope now). At the very least, the information helped them make an informed choice and keep track of impressions, preventing the "which one had the 5pm service again?" confusion.

"What Should I Do Next?" – Interaction & Notifications:
This isn't a separate page but a feature that appears contextually: - On the **Dashboard**, at the very top, the app might display a highlighted suggestion like: **" Next: Book removalist or truck this week"** with a short explanation. Possibly it shows up to 3 suggestions in order of urgency. - If the user clicks "Why?", a pop-up shows: "Peak moving season – bookings fill fast [111] . You've set mid-Jan as move time, so aim to book by early Nov." (Citing our doc). - The user can mark it done right there if they completed it outside the app, which effectively checks off the corresponding task. - There might also be a dedicated "What's Next" screen or tab where a more detailed coach view is available. It could group suggestions by domain or time frame ("This week focus on: [task1], [task2]. Coming soon: [task3] next week."). - The engine also ties into notifications: If the app is allowed to send notifications (maybe as PWA push notifications or email), it could remind the user: e.g. "Don't forget: Sylvie's NDIS address update is due" or "Your appointment tomorrow at 4pm – good luck!". - The suggestions adapt as tasks are completed. For example, once the truck is booked, that suggestion disappears and the next priority (maybe "Start decluttering garage") appears. - Another example: after moving, the engine's focus shifts. It might suggest community or follow-up tasks rather than pre-move tasks. E.g. in February: "Register for a local GP" or "Review what worked/what didn't in the move for next time" (perhaps not needed but could). - Importantly, the suggestions will never be generic template text; they use the family's actual data. If a task is overdue, it likely becomes a suggestion with maybe an alert icon. If the user snoozes or ignores suggestions, the system could escalate urgent ones (turning red, or sending a push).

The **notification logic** will prioritize health- and deadline-critical tasks. For instance, tasks like “transfer MS infusion” or “apply for WWCC” will get flagged early and repeatedly since failing those has major repercussions (health and employment, respectively). The app might implement a simple priority system: each task has a priority (High for must-do, Medium, Low) possibly set by a combination of domain (Health tasks default High) and due date proximity. The Next engine then picks highest priority + most imminent items.

In summary, the UX is designed to break down a massive undertaking into manageable pieces, presented in context. Each module flow above shows how the app might guide the user through inputting their decisions or showing them what to do, backed by the information from their own planning documents. The interface favors checklists, simple forms, and visual aids (like progress bars or calendars) to accommodate the busy, often exhausted state of the users. The ultimate measure of success is that Josh and Kristy can **open the app at any moment and quickly see what needs attention next**, without having to flip through dozens of Google Docs or keep mental tabs on everything. PDM will function like a calm project manager friend who always knows the plan.

E. Implementation Plan (Phased)

Building PDM is a substantial project, but we will approach it in 3–4 iterative phases, each delivering a usable subset of features. This phased approach lets Josh (the solo developer) start benefiting early (by tracking basic tasks) and gradually add complexity (document parsing, intelligent suggestions, etc.) with AI coding assistance. Below is the proposed roadmap:

Phase 1: Core Data & Basic Functionality (MVP)
Objective: Establish the foundation – data models, basic UI for manual task management, and a simple timeline. Enable the user to start entering and checking off tasks as soon as possible. - **Epics:** 1. **Project Setup & Auth:** Set up the project structure (React front-end and Node/Express + SQLite back-end). Implement a simple login (even if just a fixed password or local auth since it’s single-user) for security. 2. **Data Model & Migrations:** Create the database schema for key entities (User, Document, Task, Domain, Property, etc. as outlined) [122] . Ensure we can store tasks and relate them to domains at minimum. 3. **Task Management UI:** Develop pages to create, edit, and complete tasks (with domain/category tagging). Start with a generic checklist view that lists tasks grouped by domain. The user can manually add tasks and mark them done. This will be the initial “Checklist” module. 4. **Basic Timeline/Calendar:** Implement a simple timeline or calendar view that shows tasks with due dates. This could be an “agenda” list of upcoming tasks by week. For MVP, we can omit a full drag-drop calendar; just list tasks sorted by date and highlight overdue ones. 5. **Module Stubs:** Create placeholder pages or sections for each domain module (Housing, Work, etc.) – initially just showing the tasks of that category. This scaffolding prepares for adding specialized UI later. - **Acceptance Criteria:** - The app runs and stores data locally. A user can log in and see a dashboard or home screen. - The user can create a task with at least a description, due date, and category, and it appears in a list. - Tasks can be marked complete (with UI feedback like strikethrough or moving to a “Done” list). - Tasks with due dates in the past appear highlighted as overdue. - There is a rudimentary timeline view showing tasks by date. - No document upload or advanced logic yet – but the system is stable with 50+ tasks entered manually (testing the data model constraints). - Phase 1 deliverable is essentially a digital to-do list tailored to moving categories – enough to replace a static checklist doc.

Phase 2: Document Ingestion & Intelligent Task Generation
Objective: Introduce the ability to upload documents and automatically generate tasks and data from them.

This will significantly populate the app with the family's existing planning info. - **Epics:** 1. **Document Upload Backend:** Implement an `/api/documents/upload` endpoint to accept file uploads (PDF, DOCX, images) [123]. Use a library (e.g. Python `python-docx` or `pdfplumber`, or Node alternatives) to extract text content. Store the file (or its text) in the database. 2. **Document Management UI:** Create a front-end interface to upload files and list uploaded docs. After upload, show a preview or confirmation of text extracted. 3. **Task Parsing Service:** Develop a script or service that processes the extracted text to identify actionable items. This could be rules-based (looking for bullet points or numbered lists in the text) or AI-assisted (e.g. run a prompt on the text like "list all actionable tasks with deadlines"). Given time constraints, start with simple heuristics: - If a document has sections like "8 Weeks Before Move – …", parse those into tasks with relative dates. - If bullet lists contain verbs ("cancel, update, book"), consider each a task in the appropriate domain (perhaps map keywords: "insurance" -> Finance domain, "book removalist" -> Logistics domain). - The Move Checklist doc is a prime candidate to parse automatically: It's structured in categories and timeline chunks [39] [34]. We can code a parser that recognizes those headings (like the emoji categories and timeline headers) and create tasks accordingly, with due dates if relative timing is given (we know the target move date, so "8 weeks before" can be calculated). - We will likely refine this with a bit of manual tuning or an AI summarizer. The acceptance criteria can be that at least the majority of tasks from Move Checklist and maybe the Move Plan doc are captured without manual re-entry. 4. **Link Tasks to Docs:** Enhance the Task model/UI to record origin_doc and possibly a snippet. So, if a task was generated from a doc, the user can click "source" and read the context (we could show a few lines around where it was found in text). This traceability builds trust. 5. **Basic Domain Data Extraction:** If possible, parse specific structured info: e.g. from the Comparison doc, extract the chosen Option C. From the daycare doc, extract the list of recommended centers. From the neurologist doc, extract the list of doctors (populate Provider entries). This might be done via targeted regex (e.g. find lines with Dr. or addresses) or via manual copy initially. It's somewhat ambitious to fully automate, but even partially filling these tables saves time. - **Acceptance Criteria:** - The user can upload at least the provided docs (the ones we have in the corpus) and the app successfully extracts text (no crashing on complex formatting). - After upload, the system either automatically or via a "Process documents" action generates a set of tasks. Specifically, after uploading "Move Checklist.docx", the tasks from its content appear in the app (e.g. "Notify Bendigo Bank of address change", "Book removalists by mid-Jan") with correct categorization [124] [79] and approximate due times (the timeline ones placed on calendar accordingly). - Similarly, uploading "Updated Moving Plan" might populate the timeline with the Jan 13–19 events, and "Moving Options A–D" might create a note or task indicating Option C is recommended (maybe create a non-checklist info item in Housing domain like "Chosen strategy: Option C (hybrid)"). - The user should be able to review and adjust the generated tasks. All auto-tasks should be editable and deletable (in case parsing included something irrelevant). - Document text is stored for reference, and clicking a task's source lets the user see it (e.g. in a tooltip or side panel showing the lines from the doc with citation). - No AI hallucinations: ensure tasks generated truly come from the text (this might mean we favor under-generating vs. over-generating, to avoid bad data).

By end of Phase 2, the app will basically contain all the family's known tasks without them typing it all in – a big win in setup time.

Phase 3: Domain-Specific Decision Support Views
Objective: Build out the special interfaces and comparison tools for the decision-heavy modules (Rentals, Job options, Childcare, Providers). Also implement the people/vehicle matrix for logistics. This phase makes the app much more than a todo list – it becomes an interactive planner tailored to their move. - **Epics:** 1. **Housing Module UI:** Implement the full Housing & Rentals UI. This includes forms for adding/searching rental properties (with fields like address, rent, notes) and a section for Bendigo house tasks. Possibly

integrate a map view using Google Maps API (optional). The comparison might be simple table listing (address, rent, distance to DCS) or a nicer card layout. Also include fields to log overlap dates (lease start, lease end of Bendigo) to calculate overlap duration or costs. **Acceptance Criteria:** User can add a rental property, see it in a list, and mark one as "decided". Bendigo checklist tasks from earlier phases are visible here filtered to property domain. No broken links – e.g. if user marks a rental as chosen, that state persists. 2. **Kristy's Work Module UI:** Create interface to list multiple JobOption entries and compare them [125]. A simple approach: a table with one column per job and rows for criteria (Pay, Hours, Pros, Cons, Alignment score). Or a vertical list of cards. Let the user set one as chosen. **Acceptance Criteria:** At least two job options can be input and their details viewed side by side. The system maybe suggests criteria to consider (like fatigue impact) in text, but calculation of an "alignment score" can be rudimentary (e.g. number of conflicts with childcare schedule). 3. **Childcare Module UI:** Implement listing of childcare centers with relevant info (age criteria, distance, etc.) [126] [45]. Possibly integrate a small map or just addresses. Allow marking enrollment status and desired days. Also, show a combined schedule for kids. **Acceptance Criteria:** User can add centers or edit the pre-populated ones, and mark which one(s) they will use. The alternate week logic should reflect in a human-readable way (e.g. a note that "Tuesdays in Week B are free – no booking needed when Josh is off"). If Sylvie's preschool is handled similarly, ensure that can be indicated. 4. **Health/Providers Module UI:** Build a page to list providers (particularly neurologists and therapists) in a structured way [55]. Could be grouped by type (Doctors vs Therapists). Perhaps have a toggle to show "Bendigo providers (old)" vs "Dubbo providers (new)". Provide a way to log contact status or appointment date next to each. **Acceptance Criteria:** The neurologist options from the doc appear with their details. The user can mark one as selected or add a new one. They can input an appointment date which then shows up in the calendar. And they can check off tasks like "sent referral" per provider. 5. **People/Vehicle Matrix:** Develop a component (perhaps on the Logistics module) to input and display the travel assignments. This might be a specialized form with repeating rows. We can model it akin to a small spreadsheet. **Acceptance Criteria:** The user can create entries for each trip (date & from→to) and for each assign a vehicle and list of people. The UI should ensure one person can be listed in two vehicles on the same date (maybe just rely on user to not do that, or highlight duplicates). It should be printable (so likely a simple HTML table layout that prints well). If time, allow exporting to CSV or PDF. 6. **Integrations & Polishing:** By now, we have many pieces. Ensure that, for example, when a user marks a rental property as chosen, perhaps that triggers the task "sign lease" or "setup utilities" if not already done (some cross-module prompt). Or if Kristy's job choice is made, it might tie into the childcare pattern. We can add a bit of glue logic: e.g. if JobOption chosen requires 3 days/week and currently childcare is set 2 days, show a warning. Or if a rental chosen starts on a certain date, fill that into the timeline as "Lease begins". **Acceptance Criteria:** Cross-module data flows make sense. E.g., if user entered lease start/end dates in Housing, the timeline on Logistics reflects "lease overlap period". Or if they set actual move-out date, the Utilities module knows to recommend disconnection that date. Doesn't have to be fully automatic, but at least context is shared (we might store a single `move_date` in a config that all modules can read). - **Acceptance Criteria for Phase 3:** - All 11 modules now have a dedicated UI as per design (though some may still largely be a filtered task list with a bit of extra info, like the Church module might not be heavy on interactivity beyond notes). - The user can use the app to make real decisions: pick a rental, pick a daycare, choose a church, etc., and those choices persist and can be reviewed. - The people/vehicle matrix can be filled out and viewed clearly (matching what was planned on paper). - The timeline view now includes key dates and maybe can distinguish tasks vs fixed events. - The app at this stage should cover the full scope of planning – meaning Josh and Kristy could theoretically run the remainder of their move with just this tool. Phase 3 is basically feature-complete in terms of domains (the next phase will add AI guidance and refinements).

Phase 4: Smart Guidance & AI Enhancement

Objective: Layer in the intelligent assistant features: the "What Next" engine for recommendations, and any advanced advice or Q&A capabilities using the document knowledge (RAG). Also address any remaining nice-to-haves or performance tweaks. - **Epics:** 1. **"What Should I Do Next?" Engine:** Implement the logic to select top tasks. Likely a server-side function that filters tasks by status=Pending, sorts by a combination of priority and due_date, and then maybe applies some custom rules (e.g. always surface one high-priority health task, one logistics task, etc., to ensure variety). Incorporate dependencies: if a task has prerequisite tasks (we might encode these manually in the data or infer by context), avoid suggesting out-of-order things. **Acceptance Criteria:** When the user has many tasks, the dashboard (or a dedicated Next screen) shows 1-3 suggestions. The suggestions feel relevant (e.g. not suggesting a low-priority far-out task when an urgent one is undone). If user marks a suggestion done, it disappears and next one comes up. 2. **Contextual Rationale (Advice Generation):** Use Retrieval-Augmented Generation to provide reasoning or tips alongside suggestions. This could be done by storing the documents' embeddings and at suggestion time, pulling the most relevant paragraph to that task and either showing it or using an AI to rephrase it. For instance, if "book removalist" is suggested and we have a snippet "peak season… book by late winter" [10] [127], the app can display: *"(Because it's peak moving season, full-service bookings fill up fast – best to lock in by early Dec.)"*. We can implement this with a lightweight local model or an API call to OpenAI with a prompt like "Explain why this task is important now, based on context: [insert snippet]". Since this is for personal use, using an API might be fine. **Acceptance Criteria:** At least for a few test tasks, clicking "why" produces a helpful explanation that references their situation (and possibly cites a source doc line in the app UI for transparency). 3. **Natural Language Query (Stretch):** Possibly allow the user to ask a question in the app, like "When do we need to do the license transfers?" and have the app answer from the data (e.g. "Within 3 months of moving, ideally by March 2026. We have a task for it in Licensing."). This is a bonus if time permits, leveraging the same RAG setup. 4. **Offline & Performance:** Make sure the app works offline or with spotty connectivity (since the target environment might be on the road). Using a service worker to cache static files and perhaps caching the last state of data in localStorage for read-only access offline can be done. **Acceptance Criteria:** If internet is off (for a hosted version) or if running locally, the app still loads and shows the last known data. The user can at least view tasks offline; editing offline might be too much complexity unless using a PWA with IndexedDB sync, which might be future work. 5. **UX Refinements & Polish:** Clean up any rough edges from Phase 3. Improve mobile layouts (all pages should be usable on a phone screen). Add usability tweaks like drag-and-drop ordering of tasks within a category, or color-coding domains, or a progress overview. Ensure prints of checklists and plans are formatted nicely (e.g. dark background elements avoid printing or switch to print-friendly). 6. **Testing & Debugging:** Write unit tests or integration tests for critical logic (especially the Next engine and any date calculations). Manually test flows with realistic data to catch issues (like extremely long text, special characters from documents, timezone issues for dates, etc.). - **Acceptance Criteria:** - The "What Next" suggestions appear on the dashboard and seem context-aware. For example, when testing with incomplete data that mirrors 6-8 weeks before move, it suggests tasks that align with that timeframe (like booking movers, transferring Tysabri) instead of tasks due on move day (which would be suggested later). - Each suggestion has an explanation either directly visible or on click. Explanations are drawn from the user's documents or preset knowledge (not generic stock advice). Ideally, they include a reference or at least clearly relate to the family's situation, giving the user confidence. - The user can mark tasks done from the suggestion interface, and it updates the main lists accordingly (state consistency). - Overall performance is acceptable: uploading a doc and parsing it doesn't freeze the app (if needed, do it asynchronously with a spinner). The UI stays responsive even with 100+ tasks loaded. - The app can be considered "feature-complete" for the Parris family's move. Any remaining tasks or ideas are logged for future (see next section), but not required for a successful move in Jan 2026.

By following these phases, we ensure that by the end of **Phase 1** Josh can start using the app in basic form (entering tasks as he thinks of them). **Phase 2** will quickly enrich it with all the content from their meticulous planning docs – essentially migrating their brain dump into a structured system. **Phase 3** will give the real power tools to analyze and decide using that data (the comparisons and matrix scheduling that would be hard to do in a normal checklist). And **Phase 4** caps it off with intelligent guidance to handle the overload and edge cases, leveraging AI where it's most valuable.

Each phase deliverable is shippable and testable, so Josh can gradually trust the app with more and more of the plan (instead of a big bang delivery on Jan 1, which would be risky). We've front-loaded the must-haves (tasks, dates) and left nice-to-haves (AI Q&A, extra polish) to last.

F. AI Autocoder Prompts

To accelerate development, below are **15 prompts (CT1–CT15)** ready to paste into an AI coding assistant (like GitHub Copilot Chat or Cursor). Each prompt is self-contained, specifying the context, the goal, and acceptance criteria for a development task. They assume a project named `pdm-parris-dubbo-mover` with a **TypeScript React front-end** and a **Node.js (Express) + SQLite back-end**:

CT1 – Project Scaffolding
Create a new project called **pdm-parris-dubbo-mover** with a TypeScript React frontend and a Node.js + Express backend. Set up the backend to serve API requests (e.g., running on `/api` routes) and the frontend as a Single Page App. Include a basic Express route `/api/health` that returns `{status:"ok"}` for testing. The React app should have an initial Home page that displays "PDM App running". Set up a SQLite database (use an npm library like `better-sqlite3` or `sqlite3`). Ensure the project can be started with one command (concurrently run server and client). **Acceptance Criteria:** Able to run `npm start` and see Home page at http://localhost:3000 and get "ok" from http://localhost:3000/api/health. Project structure (frontend and backend folders) and a README.md explaining how to run it are present.

CT2 – Database Models & Migration
In the Node backend of **pdm-parris-dubbo-mover**, define the database schema and create migration scripts for the core models: **User, Document, Task, Domain, Property, JobOption, ChildcareOption, Provider, Appointment,** and **Trip/TripAssignment**. Use a migration tool or simple initialization script to create tables with the fields discussed (IDs, foreign keys, etc.). For example: - `User`: user_id, name, email, password (hash). - `Domain`: domain_id, name. - `Task`: task_id, user_id, domain_id, description, due_date, status, priority, origin_doc_id, related_property_id, related_job_id, related_childcare_id, related_provider_id. - (and similarly define fields for Property, JobOption, ChildcareOption, Provider, Appointment, Trip, TripAssignment). Include necessary foreign key constraints (e.g. Task.user_id -> User, Task.domain_id -> Domain, etc.). **Acceptance Criteria:** Running the migration (or starting the app) creates the SQLite database with all tables. The schema matches our design (all specified fields present, types appropriately chosen). Demonstrate by outputting the list of tables or a sample insertion (not required to implement all foreign data yet, just structure).

CT3 – User Authentication (Basic)
Implement a simple authentication system. Create an Express route `/api/login` that accepts a username/email and password, and checks against the User table. You can pre-seed the User table with one

user (Josh) with a known password (hashed, e.g. use bcrypt). For now, a successful login can start a session (use Express sessions or JWT – choose simplest). Also protect subsequent routes by checking `req.session.user` or a JWT token. Additionally, add a middleware to simulate auto-login in development (for ease, if a certain env flag is set, bypass auth). On the frontend, add a Login form component that calls `/api/login` and on success, stores the session (for JWT, store token). **Acceptance Criteria:** Hitting a protected endpoint (e.g., upcoming `/api/tasks`) without login gives 401. Logging in with correct credentials returns 200 and thereafter the client can access protected data. The login state persists (session cookie or JWT stored). It's okay to have a hardcoded user for now.

CT4 – Task API & Frontend List
Implement CRUD for tasks. On the backend: - Create routes: `GET /api/tasks` (returns all tasks for the logged-in user), `POST /api/tasks` (create new task), `PUT /api/tasks/:id` (update task fields like status or due_date), `DELETE /api/tasks/:id`. On the frontend: - Create a TaskList component that fetches `/api/tasks` and displays tasks in a list or table. Show fields: description, due_date (formatted), domain (you can show domain name by joining with Domain table in the API or sending it pre-joined). - Provide a form to add a new task (text input for description, date picker for due_date, dropdown for domain). - Allow marking a task complete: e.g. a checkbox that triggers a `PUT /api/tasks/:id` to set status = "done". - Allow deleting a task (a small "✕" button). **Acceptance Criteria:** The user can view all their tasks loaded from the server. Creating a task via the form adds it to the list (without page refresh). Marking a task done immediately updates its appearance (e.g. crossed out or moved to a separate "completed" list) and persists (confirmed via page reload or re-fetch). Deletion removes it from view and DB. Domain handling: for now, you can use a static list of domain names or fetch from `/api/domains` if implemented. Basic error handling included (e.g. form validation for required description).

CT5 – Domain Filtering & Module Navigation
Enhance the frontend to organize tasks by domain module. Implement a sidebar or menu with the 11 module names (Housing, Work, Childcare, etc.). When a user clicks a module, navigate (e.g. using React Router) to a route like `/module/housing` that shows tasks filtered to that domain. Backend: ensure the GET /api/tasks can accept a query param for domain (e.g. `/api/tasks?domain=Housing` or domain_id) to return only those tasks. Frontend: Implement ModulePage component that uses the domain param and displays relevant tasks (reuse TaskList component but filtered). Also show the module title and maybe a placeholder for future module-specific UI ("Housing module coming soon" text). **Acceptance Criteria:** Clicking each module in the UI shows only that module's tasks. The filtering works both client-side and via API (we should test that only Housing tasks come when requesting domain=Housing). The navigation should be smooth (no full reloads). This sets the stage for adding specialized components to these pages in future tasks.

CT6 – Document Upload & Parsing (Backend)
Implement an endpoint to handle document uploads and extract text. Use an NPM library (or Python via child process) to read PDF/DOCX: - Add a route `POST /api/documents` that accepts multipart form data (the file). Save the file to a `uploads/` directory (or in memory). - Use a library like `pdf-parse` for PDF and `textract` or `docx-to-text` for DOCX (or `mammoth` for docx) to extract text content. For images, you could integrate an OCR library (optional; can skip if not needed now). - Store a new Document record in the DB with title (filename) and the extracted `content_text`. - Return the document's ID and maybe the first N characters of text as confirmation. **Acceptance Criteria:** Able to upload a known DOCX (e.g. "Move Checklist.docx") and the server stores it. Verify in DB that Document entry exists with the full text (we can log the length or a snippet to console for proof). If any library fails due to format, handle gracefully (return

an error message). The endpoint responds with 200 and basic info for frontend to use. The file itself can be saved on disk but the primary goal is to have its text in the DB.

CT7 – Document Ingestion & Task Generation (Backend)
Implement logic to generate tasks from a Document's text content. Focus on the "Move Checklist" as a primary example: - After uploading a document (or via a separate route like `POST /api/documents/:id/ingest`), parse the stored text for tasks. For the checklist, perhaps split by newline and find lines that start with `*` or numbers. - You might hardcode logic for known docs: e.g., if doc title contains "Checklist", then treat lines with `*` as tasks. If it contains "Weeks Before", handle accordingly. - For each identified task line, create a Task entry. Attempt to categorize: e.g., if the line contains keywords like "electricity" or "gas", domain = Utilities; "GP" or "NDIS" -> Health; "preschool" -> Childcare, etc. (We can use a simple mapping dictionary). - If the line has a timeline indicator like "8–10 Weeks Before", we might ignore it as it's a header, not a task. For now, focus on concrete lines. - Avoid duplicates: maybe don't re-create tasks that already exist (you could skip if a similar description exists, or just assume multiple ingestion runs won't happen for same doc). **Acceptance Criteria:** When calling the ingestion after uploading Move Checklist, the system creates multiple Task records corresponding to items under categories [39] [74] [32]. For example, tasks like "Cancel or transfer utilities", "Update ATO, MyGov, Medicare…", "Declutter/donate unwanted items" [128] appear in the DB and are assigned to plausible domains (Utilities, Finance/Admin, Packing respectively). The tasks should not include the category headers themselves (no task just called " Housing & Property"). This can be tested by checking GET /api/tasks returns these new tasks. We should see on the front-end (if refreshing the task list) many new tasks populated. Accuracy of domain assignment can be rough – just ensure some categorization happened. The ingestion process should be idempotent per document or at least not crash if run twice (perhaps skip if tasks exist).

CT8 – Module: Housing & Rentals Page
Build out the Housing module UI with special features: - On the Housing module page (`/module/housing` route), in addition to the task list, add a section for **Dubbo Rentals**. Create a small form to input a new RentalProperty (address, rent, notes). Show a list of RentalProperty entries (you'll need to create a GET `/api/properties?type=dubbo_rental` and a POST `/api/properties`). - The list should display each property with key info and maybe a "status" field (e.g. applied, shortlisted, secured). Include a button or checkbox to mark one as "Chosen" (this can be a property field like `is_chosen` boolean). - Also include a subsection for **Bendigo House** tasks specifically: filter tasks where domain = Housing and maybe a certain tag like "Bendigo". Alternatively, simply show all Housing tasks (which will naturally include house prep tasks). If possible, differentiate "new home" vs "old home" tasks by checking description keywords (e.g., tasks containing "53 Buckland" or "Hutton" pertain to the old house). - Ensure that adding or updating a property refreshes the list. Marking one as chosen could, for example, disable choosing others (only one can be primary). **Acceptance Criteria:** On Housing page, user can add a rental property entry and see it immediately listed (persisted via API). Example: input "12 Church St – $570 – Note: close to park". It appears in list. The user can mark "Chosen" on one property; if another was previously chosen, ensure only one remains chosen (enforce either in UI or API). The Bendigo house tasks (like "Clean and photograph property" [129]) are visible on this page; it's clear they're part of the housing domain tasks. Essentially, housing page now combines structured data (properties list) and unstructured tasks.

CT9 – Module: Childcare & Schooling Page
Implement the Childcare module page with features for Elias's daycare and Sylvie's preschool: - Create backend routes for ChildcareOption similar to properties (GET `/api/childcare_options`, POST to add). - On `/module/childcare`, display a list of ChildcareOptions. Pre-populate (in the DB seed or via ingestion

if done) a few from our docs (Imagine, Community Kids, Goodstart, etc.). Show key info: name, type (long daycare vs preschool), and any notes (like “can take under 3”). - For each, allow an “enroll” action or selection. Perhaps a toggle “Enrolled” or a dropdown for status (Not contacted, Waitlisted, Enrolled). - Provide an interface to input the planned schedule: e.g., a small table or text that says “Elias: Mon, Tue (every week except alt Tue)” and “Sylvie: Mon, Tue at DCS (if accepted)”. This could be just static text fields for now where user can write the days, as a full scheduling UI is complex. Or simpler: a note field on each ChildcareOption where user notes the days. - Ensure tasks related to childcare (domain) are shown too (like “Update childcare subsidy in Centrelink”). **Acceptance Criteria:** The childcare page lists centers (e.g. Imagine, Community Kids from our example data) with at least their name and a short description. The user can mark one as chosen for Elias by toggling “Enrolled” on it. Maybe an icon indicates that’s Elias’s daycare. For Sylvie, if we treat DCS Preschool as a ChildcareOption too (type=preschool), the user can mark if she’s accepted there. The page should reflect the pattern: if possible, show a label like “Week A: Mon/Tue, Week B: Mon (Tue with Dad at home)” once they configure alternate Tuesdays off (this can be a static implementation: e.g., a text computed if a ChildcareOption is marked as “Every second Tuesday off”). At minimum, let the user input a note about schedule and display it. The tasks list on this page shows things like confirming enrolment or waitlist follow-ups if those tasks were ingested. Overall, the user feels this page helps visualize the kids’ care plan, not just tasks.

CT10 – Module: Health & MS Page
Build the Health & MS transition page to track providers and tasks: - Backend: ensure Provider model and routes (GET `/api/providers`, POST, PUT). Seed it with some known entries (e.g. “Dubbo Base Hospital – Neurology Clinic”, “Dr Ruhaida Daud”, etc from our list). - Frontend `/module/health`: show a list of Provider entries relevant to MS/health. Could group by category (Neurologists vs Other). Display name, location, and a status (like “to contact” or “first appt booked” – maybe use a field or derive from whether an Appointment exists for that provider). - Next to each provider, if an appointment is scheduled (check Appointments table), show the date (e.g. “Appt on Mar 23”). - Provide a button “Add Provider” to add a new one (with a small form popup for name, type, contact). - The tasks section on this page should filter to Health domain tasks (transfer GP records, get referrals, etc.) [32] . **Acceptance Criteria:** The Health page lists at least the seeded providers (e.g., "Dubbo Health Service Neurology Clinic – Dubbo Hospital – Status: not contacted"). The user can add a provider (say they got a referral to “Dr Kieren Po”) via the UI and it appears. They can edit a provider to update a note (e.g. “Appt on 5 Feb scheduled”) or we handle that by adding an Appointment: If you have time, implement a form to add an Appointment (provider dropdown, date) which saves to DB and then is displayed under that provider. At minimum, allow entering a date field on the provider itself for next appointment as a simplification. The tasks like “Organise Kristy’s infusion transfer” are visible; once the user marks those done, they disappear from this list (but remain globally). Essentially, the page combines a structured list of doctors/clinics with the checklist of health tasks.

CT11 – Module: Logistics (Packing/Travel) Page
Implement the Packing, Decluttering, Logistics & Travel module page: - This page will have a few components: - **Packing Checklist:** Filter tasks for Packing/Moving domain (things like packing rooms, first-night box, etc.) [84] [85] . Possibly group by subcategory (if tasks have keywords like “Pack” vs “Declutter”). - **Big Items Decision:** A table or list of special items (maybe derive from tasks or create a small JSON) – e.g. Spa, Trampoline, Chickens, etc – with their decided status. This can be static for now: e.g., hard-code those as part of UI with a dropdown “Taking / Selling / TBD”. Or if time, have a small array and state to store user selection. - **Trip Planner (Vehicle Matrix):** Create a section where the user can input travel plans. You can simplify data entry by focusing on the known two trips: For trip 1 (Bendigo->Dubbo on say Jan 15) and trip 2 (couple days later), provide a mini-form to enter who’s in each car. For example, a form row for “Truck:

[Driver dropdown], [Passengers multiselect/text], note". Implementing a full dynamic matrix is complex, so you could predefine 3 vehicles (Truck, Car1, Car2) and for each allow entering driver and passengers for Trip1 and Trip2 separately (two columns). If implementing dynamic, you'd use the Trip and TripAssignment tables and allow adding them as needed. - **Export/Print Button:** A button "Print Plan" that triggers `window.print()` with some print CSS to format the matrix and timeline nicely. - The timeline itself might not need a separate component here if we have a global timeline page, but you could show a condensed timeline for move week on this page as well (maybe hardcoded key dates, or pulling tasks labeled as happening on specific dates). **Acceptance Criteria:** On the Logistics page, the user sees their packing tasks (e.g., "Declutter garage – done", "Pack kids room – pending"). They also see a list of "Big Items" decisions with drop-downs or toggles (e.g. Spa: "Not taking (selling)", Trampoline: "Taking", etc.) – they can change these statuses. The vehicle matrix section allows input: e.g., Trip 1 – Truck: driver select (options from known names we can hardcode: Josh, Michael, etc.), passengers text field, Trip 2 – similarly. The user can fill those in and see the layout. It doesn't have to save to DB (could just be UI state) unless TripAssignment endpoints are implemented. The Print button generates a printer-friendly output (we can test by preview). It should include the key info: who drives what on which date, and possibly a summary of tasks or a final checklist to print for helpers.

CT12 – Dashboard with "What Next?"
Implement the Dashboard page to show an overview and the "What should I do next?" suggestions. - The backend: create an endpoint `/api/next_actions` that returns 1-3 recommended next tasks. Implement a simple logic: - Query pending tasks, order by (priority DESC, due_date ASC). - Pick top 1 high-priority (maybe health domain) and top 1 by nearest due date from different domain, etc. (Keep it simple: maybe just return the top 3 soonest due or highest priority tasks). - Alternatively, identify categories that are critical and pick one each (not required). - The endpoint can also attach a short reason for each if easily available (for now, maybe just a generic phrase or a citation from task origin if stored). - Frontend Dashboard: Display a welcome message, a countdown (calculate days until move_date if we have one stored somewhere like as an Appointment or a config constant). - Display a summary of modules progress: e.g., list each domain with number of open tasks vs total. (This can be done by an API that aggregates tasks by domain or computed on frontend by filtering tasks). - Then the **Next Actions**: show the tasks from `/api/next_actions` with a brief description and maybe a "Why?" link for each. - If "Why?" clicked: either show an inline explanation if provided by API, or pop up a modal that hits another endpoint for explanation (maybe `/api/tasks/:id/explain` that in Phase 4 would use AI). For now, you can stub it to return a static rationale or a citation from the doc if you stored one. - Allow marking a next action done right there (maybe a checkbox that calls the complete task API). **Acceptance Criteria:** The Dashboard shows something like: "68 days until Move!" (assuming a target date). It lists maybe "Housing: 5/12 tasks done, Health: 2/8 done, etc." for each category (progress overview). Below, it shows "Next Actions: 1. [Task X] – (Why?) 2. [Task Y] – (Why?)". These tasks are clearly the most urgent ones (we can test by setting one task due tomorrow and others later, and see that appears). Clicking "Why?" perhaps opens an alert or modal with a placeholder explanation ("Because this task is time-sensitive according to your plan."). Marking it done immediately reflects in the UI (it disappears from next list and maybe progress updated). The logic doesn't have to be perfect but should reasonably pick important tasks (we will fine-tune in next prompt with real rationale).

CT13 – "Why" Explanations (AI Integration)
Integrate a mechanism to provide contextual explanations for tasks (this is the RAG advice part). - This will likely involve using an AI API (OpenAI) along with our document text. - Implement a backend function that, given a task or task description, finds a related excerpt from Document.content_text that justifies it. For example, if task description contains "Mail Redirection", search Document texts for "Mail Redirection" [130]

and retrieve a sentence or two around it. - Use OpenAI API (if available) to summarize that reason. E.g., prompt: “The user is asking why this task is important: '<task desc>'. Based on the following context from their notes, provide a brief explanation: '...context snippet...'.” - Expose this via `GET /api/tasks/:id/explanation`. - Frontend: when “Why?” is clicked for a task, call this endpoint and display the returned explanation (in a popup or beneath the task). - Ensure the explanation includes either a direct reference to the context or a sensible reason. If OpenAI not available, you can fallback to a rule-based snippet: e.g., if task has due_date soon, explain it’s urgent due date; if keyword matches something known (“Tysabri” -> “to avoid a gap in MS treatment”), etc. But ideally use AI with our docs. - Keep responses concise (1-3 sentences). **Acceptance Criteria:** For a test task like “Redirect mail via Australia Post”, clicking “Why?” might yield: “Because you want to ensure all important mail reaches your new address for the next year, and setting up a mail redirection provides that coverage [73].” We should see that the explanation either includes a reference or clearly comes from the docs (like the quote about 12 months and $200 from the doc snippet). Another example: “Transfer Kristy’s Tysabri infusion” might pull an explanation about avoiding any gap in MS medication. We will verify that at least one or two tasks produce a meaningful explanation drawn from the document text (we may need to seed some mapping or ensure our Document texts are searchable on the server). The feature should degrade gracefully if no context is found (e.g. returns “Because it’s part of your plan to do this before moving.” rather than error). This meets the RAG-style advice requirement.

CT14 – Final Testing & Refinement
(No code, just instructions for testing)
Review the entire application for completeness and fix any bugs: - Test adding and completing tasks in each module. - Ensure data consistency: e.g., if a property is marked chosen on Housing page, maybe ensure a task “Sign lease” exists or something (if not, add a dummy task creation when chosen toggled, optional). - Make sure sensitive data (like docs text) isn’t exposed unintentionally in the UI beyond intended. - Optimize any slow parts: e.g., if searching documents for every “Why?” is slow, consider indexing Document text in memory on startup. - Cross-browser/cross-device check for responsive layout. - Add any missing error handling (e.g., show a message if file upload fails). - Security: ensure login is required for all API calls (maybe use a simple middleware for auth on backend). - Polish UI: maybe add module icons or improve the layout of lists (some CSS improvements). **Acceptance Criteria:** All modules pages load without console errors. The flows described in our design scenarios all work (simulate them). For instance, one can upload docs, auto-generate tasks, choose a rental, see next actions, etc., all without crashes. The app state persists correctly (tasks added remain after refresh, etc.). Prepare the app for delivery: update README with instructions to run, and perhaps a short note on what each module does. Once this is done, the PDM app is ready for the family to use in real life.

CT15 – Deployment Setup (Optional)
If self-hosting or sharing is needed: Set up a simple deployment for the app. Possibly configure it to run on a local machine or Heroku. - Ensure the build produces a single bundle (frontend built and served by backend express static in production). - Possibly include instructions or scripts for seeding the database with initial data (like the known doc-based tasks, providers, etc., so that it’s not empty on first run). - This might involve adding a `seed.js` to insert initial records if tables empty. **Acceptance Criteria:** Running `npm run build` and `npm run start:prod` serves the app with all features. The README includes steps to set up (like “upload your documents through the UI or place them in /uploads and run seed”). If deploying to a service, the necessary config (like port binding) is handled. This ensures Josh can easily run the app on his PC or a server with minimal fuss.

Note: This CT15 is optional if deployment is straightforward, but including it for completeness in guiding the AI developer.

(The above prompts guide the AI coder through building the app step by step. Each is numbered CT1–CT15 with a clear goal and success criteria.)

G. Risks, Limitations & Future Enhancements

Designing and implementing PDM comes with several **risks and limitations** to acknowledge:

- **Scope Creep & Time:** There's a risk of scope creep given the broad domains (11 modules) we're covering. As a solo developer with a move looming, time is of the essence. We mitigated this by phasing development and focusing on must-haves first (Phase 1 & 2). Still, there's a chance not all "nice-to-have" features (like a fully dynamic scheduling algorithm or polished UI for everything) will be complete before the move. The design explicitly prioritizes core functionality to reduce this risk. Sticking to the phased plan and deferring non-essentials will be important to meet the Jan 2026 deadline.

- **Data Overload & Accuracy:** With so many documents and auto-generated tasks, there's a risk of information overload or duplicating/conflicting entries (e.g., two slightly different versions of the moving plan could generate overlapping tasks). We addressed this by noting in Phase 2/3 to handle duplicates and highlight document conflicts [131]. The user should be prepared to review and clean up imported tasks. The Next engine might suggest something already done if data isn't cleaned (we have status tracking to mitigate that). Also, any parsing errors (like mis-reading a document line) could mislead planning. Testing the ingestion with known docs and adjusting patterns reduces this risk, but the user should verify critical tasks manually.

- **Data Privacy:** The app will contain sensitive personal info (addresses, health details, contacts). Since this is a self-hosted tool for Josh, we don't have external privacy concerns like third-party access, but we do need to secure the app. We included basic auth to prevent others on the network from snooping. Data is stored locally (SQLite), so protecting that file (or device) is important. If Josh uses cloud services (like uploading docs via an online version), then ensuring secure connection (HTTPS) and maybe encryption at rest would be considerations. For now, keeping it local/offline where possible mitigates major privacy risks.

- **Reliance on External APIs:** If we integrate OpenAI for explanations, there's a dependency risk: API could fail, or costs could incur. We treat the AI features as a bonus layer – the core app logic does not depend on them (the plan can be executed with or without the fancy "Why" explanations). If the API fails, the app should still function (maybe just show no explanation or a generic tip). Similarly, if we were to integrate maps or calendar APIs in the future, those come with reliability and cost considerations.

- **Usability for Tired Users:** There's a risk that despite best efforts, the app might be too complex or not intuitive in some areas, given the user's high stress and fatigue. We mitigated this by using familiar UI patterns (checklists, forms) and not overloading any single screen with too much info at once. Nonetheless, testing with the actual users (Josh and Kristy) is crucial – they should give

feedback early to tweak the UX. For example, if the Next suggestions are confusing or if a module page is hard to navigate, we'll need to simplify further (perhaps hiding advanced options behind toggles or providing a quick-start mode).

- **Completeness vs. Flexibility:** Our app is tailor-made for this specific move. A limitation is that it's not a generic moving app (which is fine, it's bespoke). But even for this move, if something unexpected comes up (say a domain we didn't anticipate, or a change in plan like timeline shift), the app must be flexible. We allowed for custom task entry in general, so the user can always add a new task that wasn't parsed. However, some modules are static (e.g., we pre-listed churches or daycare options). If, for instance, they discover a new daycare not in docs, they can add it manually (we provided that ability). The limitation is mostly that the quality of output depends on input docs – if something was missed in the docs, the app won't magically know it. Future enhancement could be a more open-ended note-taking or journaling section to catch such things.

Looking ahead, there are several **future enhancements** that could further improve PDM beyond the immediate move:

- **Integration with Live Services:** After MVP, we could integrate with real external APIs to automate some data updates. For example, tie into a real-estate listings API for Dubbo to pull rental property details (address, rent, availability) instead of manual entry. Or integrate Google Calendar or Outlook so that tasks/appointments in PDM automatically appear on the user's calendar (so Josh gets reminders on his phone) [132]. Likewise, maybe connect with map APIs to estimate travel times for each trip, or with postal API to actually submit mail redirection online. These were out of scope initially due to time and complexity, but are natural extensions if the tool is to be reused or maintained.

- **Mobile App or PWA:** Making PDM a Progressive Web App that can be "installed" on a phone would enhance accessibility. Features like offline caching (we started considering that) and push notifications for due tasks can be added. Given the on-the-go nature during the move (driving, etc.), having a fully offline-capable mobile app is useful. For instance, while driving to Dubbo, Josh might not have internet – but the app should still show his travel plan and tasks offline. We'd solidify offline mode and perhaps even consider a native app wrapper if needed for reliability.

- **Biofeedback or Routine Tracking:** This is more speculative, but since stress and health are big factors, one could integrate data like Kristy's fatigue or Josh's stress indicators. For example, if Josh uses a smartwatch that measures HRV (heart rate variability) or sleep, the app could gently adjust suggestions: e.g., "You seem very tired today, maybe focus on a light task like packing books rather than calling NDIS." [132]. This is an interesting future idea to truly personalize the AI's recommendations based on user state, but definitely beyond current scope.

- **Post-Move and Life Management:** After the move, PDM could evolve into a general family organizer. Many modules (childcare, health, finances) are ongoing concerns. We could add features like tracking monthly budgets (post-move financial adjustment), or a module to manage the new house build if they go that route (construction loan tasks were hinted in docs). The architecture is set up with domains, so new domains or repurposing existing ones (e.g., Housing domain could shift to "find permanent housing" if they plan to buy/build in Dubbo) is feasible. Future epics might include "House Build Tracker" or "Job Career Planner" (there was a career roadmap doc as well).

- **Multi-User Sharing:** Currently, we assume one user (or a couple sharing one login). In future, making it multi-user with role-based access could be great. For example, inviting a parent helper into just the Logistics module so they can see the travel plan and tasks assigned to them. Or sharing the Church exploration module with a friend in Dubbo to get their input. That would involve more robust auth and data separation by user, which our data model can handle with user_id fields, but the UI would need to support multiple users collaborating (syncing updates, etc.).

- **AI-driven Data Entry and Chat:** We focused on structured output, but another enhancement is an AI chat interface: the user could ask in natural language, "What are we forgetting?" or "Summarize our move plan in 5 bullet points." The AI could use the knowledge base to answer. Or allow the user to forward an email (like a quote or an update from a realtor) to the app and have it parse and add relevant info automatically. These would make the tool more interactive and reduce manual input further.

- **Error Handling and Maintenance:** As an enhancement, adding comprehensive logging (so if something goes wrong, it's easy to debug) and perhaps a way to export all data (to CSV or JSON) in case Josh wants to keep a record or switch tools. This ensures longevity of the information beyond the app itself.

In conclusion, PDM – ParrisDubboMover – is designed to be a pragmatic, user-centered solution for a one-time complex project. Its modular architecture and data-driven core set the stage for a successful move with reduced stress. By being deeply tailored to Josh and Kristy's documented plans and constraints, it serves as a reliable second brain during the moving chaos. And while it should deliver on the immediate needs (through January 2026), it's built with enough flexibility and forward-thinking that it can grow and continue to be useful as their life in Dubbo unfolds. The planned phases and prompts aim to incrementally achieve this vision, leveraging AI where appropriate to speed up development and provide smart assistance, all while keeping the human users – a busy, tired family – squarely in focus.

---

1 2 3 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 38 40 110 127 84 Comparison of Moving Options from Bendigo to Dubbo (Options A–D)
https://docs.google.com/document/d/1mOd-hcxitv6TtKdMQoDYQFfTfoY-43qdIrs6H8MvXqk

4 5 83 Deep-Dive Research Report – "Josh's Integrated Topic Map (v2)"
https://docs.google.com/document/d/1TBc63meSCSH5pSyBHc6PssA2AYz3Wcw5rv7MKU9NNSk

24 81 82 83 84 85 86 87 88 89 90 moving thinking
https://docs.google.com/document/d/1_SKmss0EsxiIZ_8qiUrdLcDSKAReRpOwSih_M_2nfC8

25 26 27 28 29 30 31 91 92 93 98 99 100 _Updated Moving Plan by Josh and Kristy - 20th Nov 2025
https://docs.google.com/document/d/1EdApRthpygnyyYg_0tpPmaf2FUgR3myPSE4rFsgAJtQ

32 33 34 35 39 52 60 61 66 69 70 71 72 73 74 75 76 77 78 79 80 94 95 96 97 101 102 108 109 111 112 113 114 116 117 124 128 129 130 Move Checklist
https://docs.google.com/document/d/1u_fVNXjlwBWeTXxMl0AelE4nsUAoQm7iS4HgexL62AY

36 37 43 65 67 68 122 123 125 131 132 ParrisDubboMover App - PDM App
https://docs.google.com/document/d/1gwgkG5cIBqyC644pP8AdtO5NE0Y-D0VCPFDnMyXDZoQ

41 42 44 45 46 47 48 49 50 51 53 54 126 _Elias Dubbo Mon/Tues Daycare Options
https://docs.google.com/document/d/1oYZbYKOn1Oop5pxCJcomGEXi5ER3n_0GJ2z33b10Moo

55 56 57 58 59 115 Neurologists Dubbo for Kristy MS
https://docs.google.com/document/d/1bZ0M-aeCKLVfgBWHSawydgw_eXceE4OXpU9cNHoVZRo

62 63 64 118 119 120 121 Internal Workings of Dubbo Christian School (DCS) and Its Associated Campuses
https://docs.google.com/document/d/1dOM0pKli25vWU-aQP9RK0u_BMMpgKJeGzw0FDTcJwBI

103 104 105 106 107 82 Dubbo Protestant Network- Hubs and Bridges
https://docs.google.com/document/d/104YkK8RBpuVSY3Lp34NLq0tYafl6G7Fl1y_FkTrG0oQ