# PlannerDashboard & VendorDashboard Documentation

📝 Creating the PlannerDashboard Schema  
  
Purpose  
The PlannerDashboard schema is designed to store all operational data for a planner, separate from authentication/profile data.  
It allows your frontend dashboard to quickly display:  
- Event counts by status  
- Upcoming deadlines  
- Notifications  
- Revenue and payment summaries  
  
Step 1: Identify the Planner Data Layers  
We first broke down what a planner dashboard needs:  
1. Events  
 - Each event has a title, description, startDate, endDate, status (pending, booked, ongoing, completed, cancelled), assigned vendors, and tasks.  
2. Tasks / Deadlines  
 - Each event can have tasks with title, dueDate, and status (pending/done).  
 - We also created an upcomingDeadlines array for quick access to deadlines across events.  
3. Notifications  
 - Embedded array storing messages for the planner with type, read/unread status, and timestamp.  
4. Dashboard Metrics  
 - Counters for quick stats: pendingRequests, bookedEvents, ongoingEvents, completedEvents, cancelledEvents.  
5. Revenue / Payment Tracking  
 - Each event can have multiple payments with amount, status, and method.  
 - Dashboard-level totals: totalRevenue (sum of paid amounts) and pendingRevenue (sum of pending payments).  
  
Step 2: Structure in Mongoose  
We used embedded arrays for events, tasks, deadlines, and notifications to keep all relevant data inside one document per planner.  
  
Pseudo-code Representation:  
PlannerDashboard {  
 planner: ObjectId (ref Planner)  
 events: [  
 {  
 client: ObjectId (ref Client)  
 title: String  
 description: String  
 startDate: Date  
 endDate: Date  
 status: Enum[pending, booked, ongoing, completed, cancelled]  
 vendors: [ObjectId ref Vendor]  
 tasks: [ { title, dueDate, status } ]  
 payments: [ { amount, status, method } ]  
 timestamps  
 }  
 ]  
 notifications: [ { message, type, read, createdAt } ]  
 upcomingDeadlines: [ { event, taskTitle, dueDate, status } ]  
 pendingRequests: Number  
 bookedEvents: Number  
 ongoingEvents: Number  
 completedEvents: Number  
 cancelledEvents: Number  
 totalRevenue: Number  
 pendingRevenue: Number  
 timestamps  
}  
  
Step 3: Auto-update timestamps  
We added a pre('save') hook in Mongoose so updatedAt is automatically updated whenever the dashboard document changes.  
  
Step 4: Benefits of this design  
- Single document per planner → frontend can fetch all dashboard data in one API call.  
- Embedded arrays → keeps related data together (events, tasks, notifications).  
- Expandable → can easily add new fields like client feedback, assigned vendors, or more financial metrics later.  
- Quick counters → metrics like pendingRequests or totalRevenue are precomputed for fast dashboard rendering.  
  
---  
  
Planner Dashboard Module Documentation  
  
Overview  
The Planner Dashboard module is part of the ElitePlan system. It provides planners with a centralized interface to monitor their events, revenue, payments, deadlines, and notifications.  
Built with:  
- Node.js & Express (backend)  
- MongoDB/Mongoose (data storage)  
- Helpers for reusable computation  
- Controllers to handle API requests  
- Routes to expose endpoints to the frontend  
  
Folder Structure  
src/  
 controllers/  
 plannerController.js  
 helpers/  
 planner/plannerHelpers.js  
 models/  
 dashboard/plannerdashboard.model.js  
 routes/  
 planner.routes.js  
  
---  
  
🧾 Vendor Dashboard Module — Documentation  
  
Overview  
The Vendor Dashboard manages each vendor’s operational metrics, orders, payments, notifications, and ratings within the ElitePlan system.  
It provides structured visibility into a vendor’s performance, financials, and client interactions.  
  
Key Components  
- Models: vendordashboard.model.js  
- Helpers: vendorHelpers.js  
- Controllers: vendordashboard.controller.js  
- Routes: vendordashboard.routes.js  
  
Core Functionality  
- Orders: Tracks client orders with statuses and payments.  
- Revenue: Calculates total and pending revenue dynamically.  
- Ratings: Stores client feedback and computes average rating.  
- Notifications: Vendor-specific messages and alerts.  
- Metrics: Counts of orders by current status (pending, ongoing, completed, cancelled).  
  
---  
  
Advantages of this Design  
- Centralized calculations ensure consistent and reusable logic.  
- Clean controller structure with minimal business logic.  
- Frontend receives ready-to-render metrics, revenue, deadlines, and notifications.  
- Scalable structure for future analytics and integrations.