# Planner & Vendor Dashboard Documentation

## Creating the PlannerDashboard Schema

The PlannerDashboard schema is designed to store all operational data for a planner, separate from authentication/profile data. It allows the frontend dashboard to quickly display event counts, deadlines, notifications, and revenue summaries.

### Step 1: Identify the Planner Data Layers

We first broke down what a planner dashboard needs: Events, Tasks/Deadlines, Notifications, Metrics, and Revenue.

Each event includes title, description, startDate, endDate, status (pending, booked, ongoing, completed, cancelled), assigned vendors, and tasks. Tasks contain title, dueDate, and status (pending/done).

The dashboard also contains notifications (messages with type, read/unread status), metrics for quick stats, and revenue tracking for both total and pending amounts.

### Step 2: Structure in Mongoose

We used embedded arrays for events, tasks, deadlines, and notifications to keep all related data within one document per planner.

Pseudo-code Representation:  
  
PlannerDashboard {  
 planner: ObjectId (ref Planner)  
 events: [ { client, title, description, startDate, endDate, status, vendors, tasks, payments } ]  
 notifications: [ { message, type, read, createdAt } ]  
 upcomingDeadlines: [ { event, taskTitle, dueDate, status } ]  
 pendingRequests, bookedEvents, ongoingEvents, completedEvents, cancelledEvents,  
 totalRevenue, pendingRevenue  
}

### Step 3: Auto-update timestamps

We implemented a pre('save') hook in Mongoose so that the updatedAt field refreshes automatically whenever the dashboard document changes.

### Step 4: Benefits of this Design

• Single document per planner — the frontend can fetch all dashboard data in one API call.  
• Embedded arrays — keeps related data (events, tasks, notifications) together.  
• Expandable — easy to add fields like feedback or vendor tracking later.  
• Quick counters — precomputed metrics ensure fast dashboard rendering.

## Planner Dashboard Module Documentation

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.

### Folder Structure

src/  
 controllers/plannerController.js → Handles HTTP requests  
 helpers/planner/plannerHelpers.js → Core functions to recalc metrics  
 models/dashboard/plannerdashboard.model.js → Schema definition  
 routes/planner.routes.js → API routes

### Helpers (plannerHelpers.js)

Helper functions perform computational logic:  
  
• updateDashboard(dashboard) — recalculates metrics, revenue, deadlines.  
• addPayment(dashboard, eventId, payment) — adds payment and triggers recalculation.

### Controller (plannerController.js)

Handles HTTP requests and updates dashboard data dynamically. Core functions include:  
• getDashboard(req, res)  
• updateEventStatus(req, res)  
• addPaymentController(req, res)  
• addNotification(req, res)

### Routes (planner.routes.js)

Endpoint Method Controller Description  
/dashboard GET getDashboard Fetch dashboard  
/dashboard/events/:eventId PATCH updateEventStatus Update event status  
/dashboard/events/:eventId/payments POST addPaymentController Add payment  
/dashboard/notifications POST addNotification Add notification

### Flow of Data

React Frontend → Express Route → Controller → Helper → Mongoose Model → MongoDB → Frontend receives updated dashboard.

### Advantages of this Design

• Centralized calculations via helpers ensure consistent logic.  
• Clean controller code for maintainability.  
• Frontend-ready metrics and summaries in one JSON response.  
• Scalable and easy to extend with analytics or vendor tracking.

## Vendor Dashboard Module — Documentation

The Vendor Dashboard is a central module managing vendor operations — including orders, payments, ratings, notifications, and performance metrics — within the ElitePlan ecosystem.

### Folder Structure

src/  
 models/dashboard/vendordashboard.model.js  
 controllers/dashboard/vendordashboard.controller.js  
 helpers/vendor/vendorHelpers.js  
 routes/dashboard/vendordashboard.routes.js

### Core Functionality

The Vendor Dashboard manages: Orders, Revenue, Ratings, Notifications, and Metrics.

### Model Overview

Fields:  
• vendor (ref Vendor)  
• orders (with client, event, status, payments)  
• notifications  
• ratings  
• averageRating  
• totalRevenue, pendingRevenue  
• pendingOrders, acceptedOrders, ongoingOrders, completedOrders, cancelledOrders

### Helper Logic

updateVendorDashboard(dashboard):  
• Recalculates all metrics whenever orders, payments, or ratings are modified.  
• Ensures real-time synchronization of data.

### Controller Overview

• getVendorDashboard — Fetch vendor’s complete data  
• updateOrderStatus — Change order state  
• addPayment — Append payment to order  
• addNotification — Add vendor message  
• addRating — Add or update client rating

### Routes

Base: /api/vendor/dashboard  
  
GET / → getVendorDashboard  
PATCH /orders/:orderId → updateOrderStatus  
POST /orders/:orderId/payment → addPayment  
POST /notifications → addNotification  
POST /ratings → addRating

### Summary

• Model — defines structure for vendor performance data.  
• Helper — manages metric recalculations.  
• Controller — implements logic and API handling.  
• Route — connects endpoints to logic layers.  
  
Together, they form a modular, scalable, and maintainable backend structure for ElitePlan.