PIDN (Patient Identifier Number) Assignment

1. **PIDN (Patient Identifier Number) Assignment**

- **Purpose:**
  - The PIDN appears to be an **internal, application-specific unique identifier for a patient record within your system**, particularly within a given organization.
  - It's distinct from an MRN (Medical Record Number), which is usually an identifier from an external EMR/HIS system.
  - Its main purpose is likely to provide a stable, unique key for patient records created and managed *by RadOrderPad itself*, especially for "temporary patients" created during order finalization before they might be linked to a more formal EMR record.
  - It helps avoid relying solely on potentially non-unique combinations like name + DOB for internal referencing.
- **How it gets assigned:**

Looking at src/services/patient.service.ts, specifically the createTemporaryPatient function:

```
// src/services/patient.service.ts

async createTemporaryPatient(client: PoolClient, organizationId: number, patientInfo: PatientInfo): Promise<number> {

 const patientResult = await client.query(

  `INSERT INTO patients

  (organization_id, pidn, first_name, last_name, date_of_birth, gender, mrn, created_at, updated_at)

  VALUES ($1, $2, $3, $4, $5, $6, $7, NOW(), NOW())

  RETURNING id`,

  [

   organizationId,

   `P-${Date.now()}`, // <—— PIDN ASSIGNMENT

   patientInfo.firstName,

   patientInfo.lastName,

   patientInfo.dateOfBirth,

   patientInfo.gender,

   patientInfo.mrn || null

  ]

 );

 return patientResult.rows[0].id;

}
```

- **Assignment Mechanism:** The PIDN is generated as P-${Date.now()}. This creates a string prefixed with "P-" followed by the current Unix timestamp in milliseconds.
- **Uniqueness:** While Date.now() is highly likely to be unique for concurrent requests (especially if they hit different milliseconds), it's not a cryptographically guaranteed UUID. For very high-throughput systems creating many patients simultaneously, there's a theoretical, albeit tiny, chance of collision if two patient creations happen in the exact same millisecond *and* the database

transaction commits them in a way that doesn't catch a uniqueness constraint (if one were placed on pidn per organization). However, for most practical purposes in this context, it serves as a reasonably unique identifier.

- **Scope:** The PIDN is generated when a *temporary patient* record is created. This happens during order finalization (PUT /api/orders/{orderId}) if the isTemporaryPatient flag is true and patientInfo is provided in the payload (as seen in src/services/order/finalize/transaction/execute-transaction.ts).
- **When it gets assigned:**
  - The PIDN is assigned **at the moment a new "temporary" patient record is created in the patients table within the PHI database.**
  - This occurs during the finalization of an order (PUT /api/orders/{orderId}) if the order is associated with a patient who doesn't yet have a permanent record in your system (indicated by isTemporaryPatient: true in the finalization payload).

2. **Order Number (order_number) Assignment**

- **Purpose:**
  - The order_number is an **application-specific identifier for an order**.
  - It provides a human-readable or system-trackable number for referencing specific orders, distinct from the auto-incrementing primary key id of the orders table.
  - It can be used in UIs, reports, and communications.
- **How it gets assigned:**

Looking at src/services/order/validation/draft-order.ts, specifically the createDraftOrder function:

```
// src/services/order/validation/draft-order.ts
export async function createDraftOrder(
  dictationText: string,
  userId: number,
  patientInfo: PatientInfo,
  radiologyOrganizationId?: number
): Promise<number> {
  // ... (user lookup, patientId extraction) ...
  const radOrgId = radiologyOrganizationId || 1;

  const orderResult = await queryPhiDb(
    `INSERT INTO orders
    (order_number, referring_organization_id, radiology_organization_id,
    created_by_user_id, status, priority, original_dictation, patient_id)
    VALUES ($1, $2, $3, $4, $5, $6, $7, $8)
    RETURNING id`,
    [
      `ORD-${Date.now()}`, // <—— ORDER_NUMBER ASSIGNMENT
      user.organization_id,
      radOrgId,
      userId,
```

```
    OrderStatus.PENDING_VALIDATION,
    OrderPriority.ROUTINE,
    dictationText,
    patientId
  ]
 );
 // ... (order history logging) ...
 return orderId;
}
```

- IGNORE_WHEN_COPYING_START
  content_copy download
  Use code with caution. TypeScript
  IGNORE_WHEN_COPYING_END
- **Assignment Mechanism:** The order_number is generated as ORD-${Date.now()}. Similar to the PIDN, this creates a string prefixed with "ORD-" followed by the current Unix timestamp in milliseconds.
- **Uniqueness:** Like the PIDN, this method provides a high likelihood of uniqueness but isn't a guaranteed UUID. If a UNIQUE constraint were placed on order_number, the database would prevent duplicates. Without it, collisions are theoretically possible but rare.
- **When it gets assigned:**
  - The order_number is assigned **when a new draft order is first created.**
  - This happens during the initial call to the validation endpoint (POST /api/orders/validate) if no orderId is provided in the request. The handleValidationRequest function in src/services/order/validation/handler.ts calls createDraftOrder in this scenario.
  - At this point, the order's status is set to OrderStatus.PENDING_VALIDATION.

Feature	Assignment Mechanism	When Assigned	Purpose
PIDN	P-${Date.now()}	During order finalization (PUT /api/orders/{orderId}) if isTemporaryPatient is true and patientInfo is provided. Creates a new record in patients table.	Internal, application-specific unique ID for patient records created by RadOrderPad, especially temporary ones.
Order Number	ORD-${Date.now()}	On the first call to POST /api/orders/validate for a new dictation (when no orderId is supplied). Creates a new draft record	Application-specific identifier for an order, used for tracking and display.

Both identifiers use a timestamp-based approach for generation, which is simple and generally effective for creating unique-enough IDs for their respective contexts within the application. If absolute global uniqueness or more robust collision avoidance were required (e.g., for distributed systems or extremely high concurrency), UUIDs would be a more standard choice.