Software Engineering II
Semester 2025-II

Workshop No. 2 — Design Artifacts and System Modeling

Miguel Angel Buitrago Castillo
Project: VetTrack

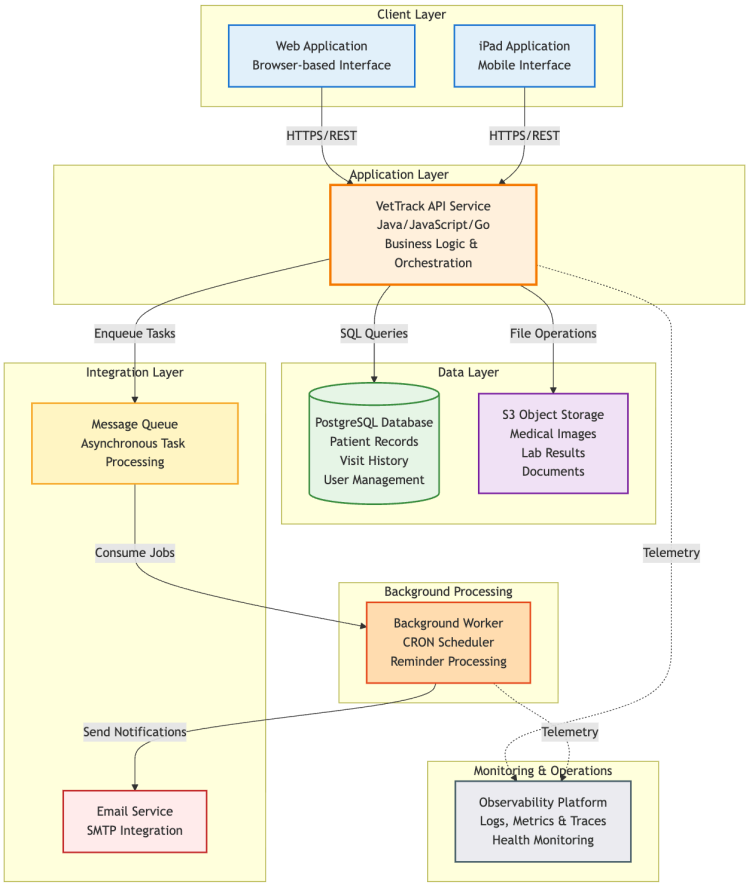
Introduction

VetTrack is a veterinary patient management system designed to support small clinics and independent veterinarians in maintaining organized, accurate, and up-to-date patient records. The system facilitates quick access to essential patient information such as medical history, recent visits, treatments, medication schedules, and follow-up needs. By centralizing this information in a clear and structured interface, VetTrack improves workflow efficiency, reduces manual record-keeping errors, and supports faster decision-making during consultations.

One of the common issues in small veterinary practices is that patient information tends to be scattered, some data is on paper forms, some in spreadsheets, some in WhatsApp chats, and some is only remembered by the vet. This makes it harder to track ongoing treatments or recall when a patient was last seen. VetTrack aims to make this process straightforward: the veterinarian logs in, sees the list of active patients, and has all the relevant details in one place. This not only speeds up daily work but also reduces mistakes and helps the clinic maintain a more consistent and professional workflow.

In its initial scope, VetTrack focuses on a single clinic environment and involves two primary user roles: the **Veterinarian**, who interacts directly with patient records and clinical workflows, and the **Admin**, who manages user accounts and basic system settings. While compact in design, the system lays the foundation for future expansion, such as support for multiple clinics, appointment scheduling, and automated reminders, ensuring that VetTrack can evolve alongside the needs of its users.

Architecture Diagram



VetTrack is built on a straightforward, reliable architecture that supports the core needs of small veterinary clinics while remaining easy to maintain and extend over time.

How Users Interact with the System Veterinarians and administrators access VetTrack through a web browser or iPad application. This gives them the flexibility to review patient records, update treatment information, add exams results, all of this while moving between examination rooms.

The Backend Service The VetTrack API serves as the central component that handles all requests from the client applications. Built using Java, JavaScript, or Go, this backend service manages three key responsibilities:

- **Storing Patient Data**: All patient information, medical histories, visit records, treatments, physical exams, and medication schedules are saved in a PostgreSQL database. This ensures that data is organized, searchable, and always available when needed.
- **Managing Patient Files**: Documents like lab results, X-rays, and other medical images are stored separately in S3 object storage. This keeps the database lean while providing secure and reliable access to important files.
- **Handling Background Tasks**: Some operations, like sending reminder notifications, don't need to happen immediately. The API sends these tasks to a message queue,

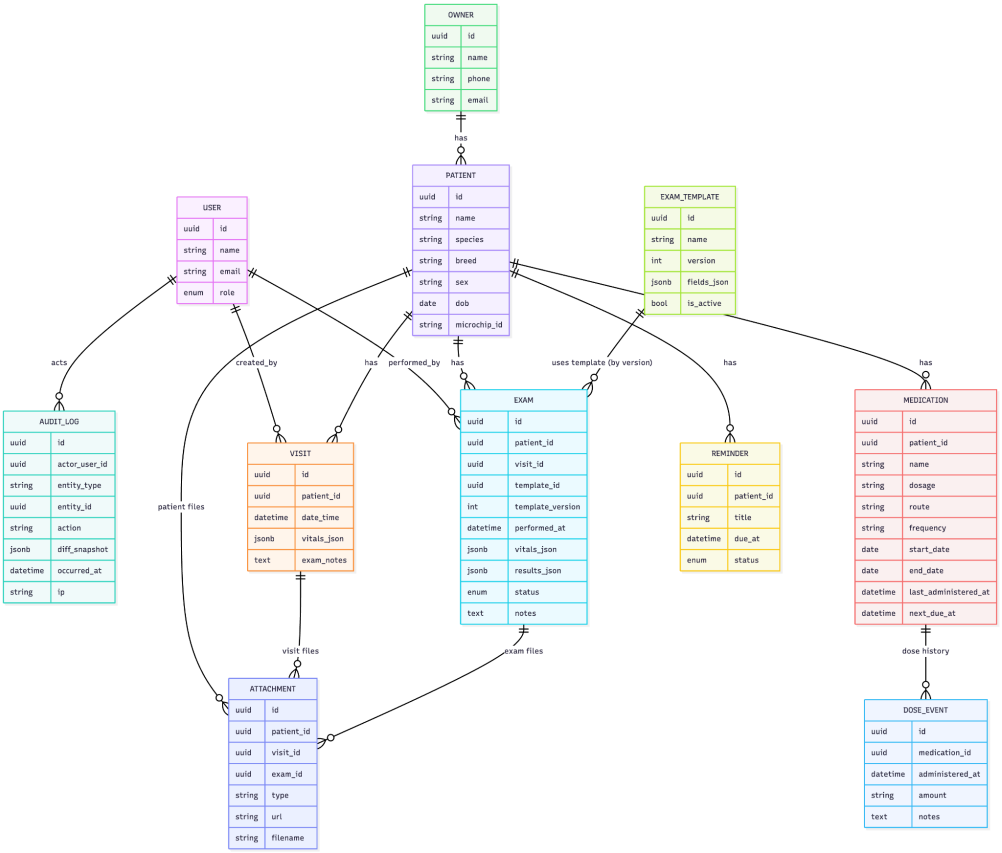
where they're processed by a separate background worker without slowing down the main application.

Background Worker and Notifications The background worker runs scheduled jobs, such as checking for upcoming appointments or medication reminders. When it's time to notify a veterinarian or pet owner, the worker sends the information through an email service. This separation means that the main application remains fast and responsive, even when processing reminders for multiple patients.

Monitoring and Reliability An observability service keeps track of how the system is performing. It logs errors, monitors uptime, and helps identify issues before they affect users. This is particularly important for a small clinic where downtime could disrupt daily operations.

Why This Architecture Works for VetTrack This design was chosen to keep things simple and focused. Each component has a clear purpose, making the system easier to maintain and debug. As VetTrack grows, potentially adding features like multi-clinic support or appointment scheduling. This architecture provides a solid foundation that can expand without requiring a complete rebuild. The goal is to support veterinarians in doing their work efficiently, without the technology getting in the way.

Domain Model



The VetTrack data model is organized around the core entities that veterinarians work with daily: patients, their owners, medical visits, examinations, medications, and related documentation. The design reflects how information naturally flows in a veterinary practice, making it intuitive to store, retrieve, and track patient care over time.

Core Entities

At the center of the system is the **Patient** record, which contains essential identifying information such as name, species, breed, sex, date of birth, and microchip ID. Each patient is linked to an **Owner**, who provides their contact information for communication and billing purposes. This relationship allows the clinic to quickly identify who to contact regarding a patient's care.

Visit and Examination Records

Every time a patient comes to the clinic, a **Visit** record is created. Visits capture the date and time, vital signs (stored as flexible JSON data), and general examination notes. Within a visit, veterinarians can perform one or more structured **Exams** using predefined templates.

Exam Templates allow the clinic to standardize how different types of examinations are recorded—whether it's a routine checkup, a surgical procedure, or a diagnostic test. Each template defines specific fields that need to be filled out, and the system tracks template versions to ensure historical exams remain interpretable even if templates are updated over time. Exam records store both vitals and results as flexible JSON structures, accommodating the varying data needs of different examination types.

Medications and Dose Tracking

The **Medication** entity tracks ongoing treatments for each patient, including the drug name, dosage, route of administration (oral, injection, topical, etc.), and frequency. The system calculates when the next dose is due based on the schedule, helping veterinarians stay on top of treatment plans.

Each time a medication is administered, a **Dose Event** is recorded with the exact time, amount given, and any relevant notes. This creates a complete audit trail of the patient's medication history, which is crucial for monitoring treatment effectiveness and identifying potential issues.

Files and Documentation

Throughout a patient's care, various documents and images need to be stored: lab results, X-rays, consent forms, and other medical records. The **Attachment** entity handles these files and can be associated with a patient generally, with a specific visit, or with a particular exam. This flexibility ensures that files are organized logically and easy to find when reviewing a patient's history.

Reminders and Follow-ups

The **Reminder** entity helps veterinarians stay organized by tracking follow-up appointments, vaccination due dates, medication refills, and other time-sensitive tasks. Each reminder is linked to a specific patient and includes a due date and status (pending, completed, or dismissed), making it easy to see what needs attention.

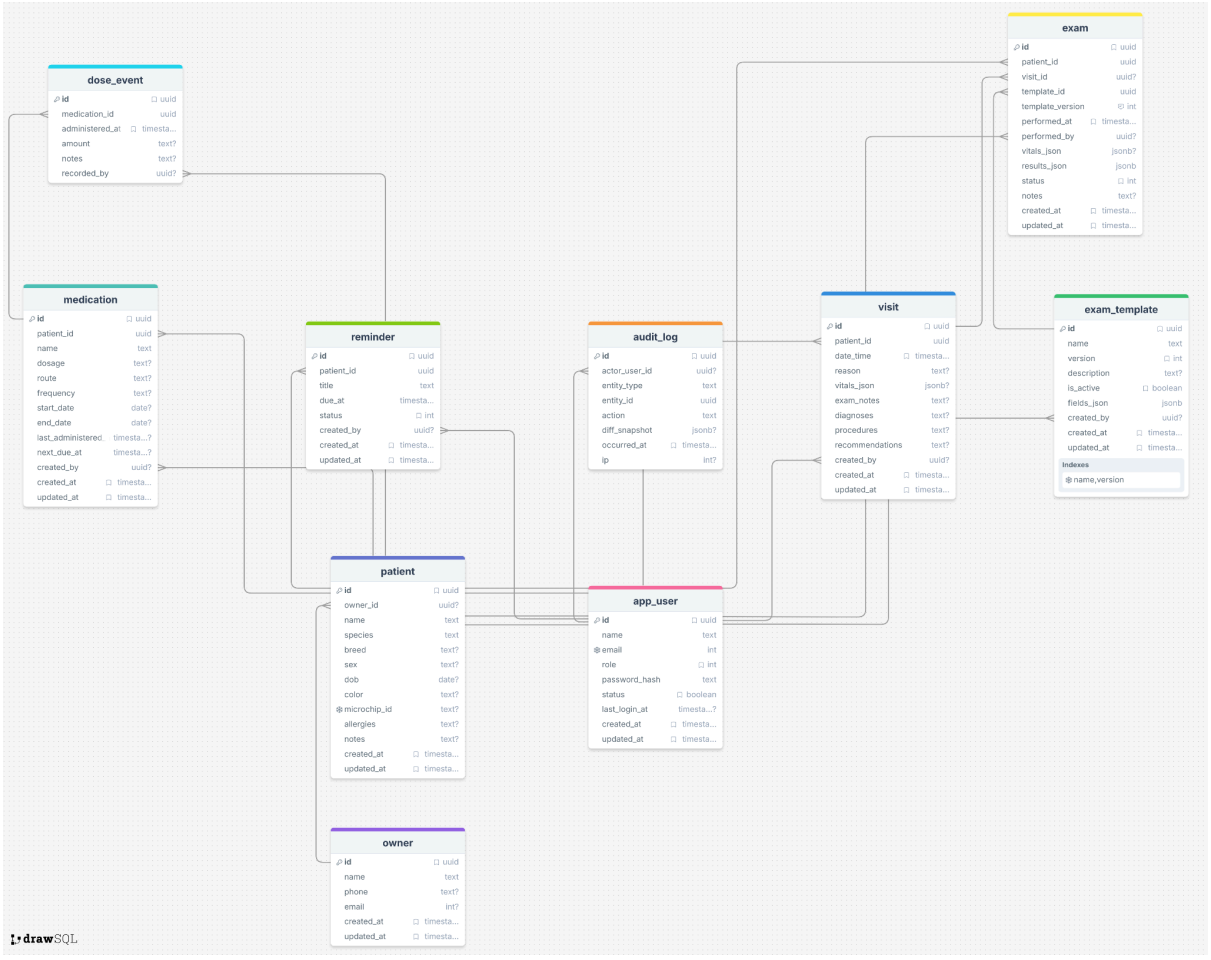
User Management and Audit Trail

Users in the system have defined roles (such as Veterinarian or Admin) that determine their permissions. All user actions that modify important data: creating visits, updating patient records, changing medications, are logged in the **Audit Log**. This log captures who performed the action, what was changed, when it happened, and from which IP address. This audit trail is essential for accountability, troubleshooting, and maintaining data integrity in a clinical environment.

Design Principles

The data model balances structure with flexibility. Core patient and visit information follows a strict schema to ensure consistency and searchability, while clinical data like vitals and exam results use JSON fields to accommodate the diverse and evolving needs of veterinary medicine. This approach allows VetTrack to be both reliable for everyday use and adaptable as clinic workflows and requirements change over time.

Relational Database Model



The VetTrack database schema implements the data model described above, with all core entities represented as PostgreSQL tables. The schema uses UUIDs for primary keys to ensure global uniqueness and facilitate future scaling scenarios.

Key Implementation Details

The database structure follows the relationships outlined in the domain model closely. Each patient is linked to an owner, and patient records connect to visits, medications, reminders, and exams. The exam system uses a template-based approach with versioning support, allowing clinics to standardize their examination procedures while maintaining flexibility.

Notable Design Choices

- **Flexible Data Fields**: Clinical data that varies by context, such as vitals and exam results, is stored as JSONB fields, providing both structure and adaptability without requiring schema migrations for each new data point.
- **Audit Trail**: The `audit_log` table tracks all significant data changes, recording the user who made the change, what was modified, and when it occurred. This supports accountability and helps troubleshoot issues.
- **Timestamps**: All entities include `created_at` and `updated_at` timestamps (with timezone support) to track when records were created and last modified.
- **User Management**: The `app_user` table stores authentication details including password hashes, roles, and account status, with last login tracking for security monitoring.
- **Medication Tracking**: The medication table calculates and stores `next_due_at` timestamps to support automated reminders, while the `dose_event` table maintains a complete history of administered doses.

The complete DDL is available in the `VetTrack.sql` file, which can be used to initialize the database structure for new installations or development environments.

CRC Cards (Class-Responsibility-Collaborator)

A) Core Domain Entities

1. User

Responsibilities:

- Represent system users; authenticate; authorize by role
- Track login activity and account status

Collaborators:

- AuthService, AuditLog, all Services (for created_by/performed_by)

---

2. Owner

Responsibilities:

- Hold guardian contact data for patients
- Enable communication with pet owners

Collaborators:

- Patient, OwnerService, PatientService

---

3. Patient

Responsibilities:

- Hold animal identity & demographics (including allergies, color)
- Expose aggregate getters (last visit, active meds, upcoming reminders)
- Serve as aggregate root for all medical records

Collaborators:

- Owner, Visit, Medication, Reminder, Exam, Attachment
- PatientService, OwnerService

---

4. Visit

Responsibilities:

- Capture a consultation (reason, vitals snapshot, notes)
- Record diagnoses, procedures, and recommendations
- Link to visit-specific attachments and exams

Collaborators:

- Patient, User (creator), Attachment, Exam
- VisitService, ExamService, AttachmentService

---

5. Medication

Responsibilities:

- Track prescribed treatment (dosage, frequency, period)
- Compute next dose due date
- Link to dose administration history

Collaborators:

- Patient, DoseEvent, User
- MedicationService, NotificationService

---

6. DoseEvent

Responsibilities:

- Log each administered dose
- Create audit trail for medication compliance

Collaborators:

- Medication, User, MedicationService

---

7. Reminder

Responsibilities:

- Represent follow-up tasks; maintain status state machine (pending → completed/dismissed)
- Support notification scheduling

Collaborators:

- Patient, User, ReminderService, NotificationServi

---

8. Attachment

Responsibilities:

- Metadata for files (images/PDFs/videos)
- Link to Patient and optionally Visit or Exam (XOR constraint)
- Track upload information and file location

Collaborators:

- Patient, Visit, Exam, User
- AttachmentService, ObjectStorage

---

9. ExamTemplate

Responsibilities:

- Define exam schema (fields/validation) with versioning
- Never mutate a version; support creating new versions
- Validate exam results against template schema

Collaborators:

- Exam, TemplateService, AuditService

---

10. Exam

Responsibilities:

- Instance of an exam; store vitals/results JSON validated against template
- Track status (draft/finalized); always linked to a visit
- Manage exam-specific attachments

Collaborators:

- Patient, Visit (required), ExamTemplate, Attachment, User
- ExamService, TemplateService, AttachmentService, VisitService

---

11. AuditLog

Responsibilities:

- Immutable record of who did what & when (entity, action, diff)
- Support compliance and debugging

Collaborators:

- All Services, Use

B) Application Services / Use-Case Controllers

12. AuthService

Responsibilities:

- Login/logout; token/session management
- Password reset and change
- Role-based permission checks

Collaborators:

- User, UserRepository, EmailGateway, AuditService

---

13. OwnerService

Responsibilities:

- CRUD operations for pet owners
- Search and filter owners
- Validate contact information

Collaborators:

- OwnerRepository, PatientRepository, AuditService

---

14. PatientService

Responsibilities:

- CRUD patients; search/filter
- Patient summary aggregation
- Coordinate owner relationships

Collaborators:

- PatientRepository, OwnerRepository, OwnerService
- VisitRepository, MedicationRepository, ReminderRepository, AttachmentRepository
- AuditService, SearchService

---

15. VisitService

Responsibilities:

- Create/update visits; validate vitals
- Attach files; link exams
- Ensure patient exists before creating visit

Collaborators:

- VisitRepository, PatientService, AttachmentService, ExamService, AuditService

---

16. MedicationService

Responsibilities:

- Prescribe/update meds; compute next dose
- Record DoseEvent; track medication history

Collaborators:

- MedicationRepository, DoseEventRepository, PatientService, AuditService

---

17. ReminderService

Responsibilities:

- CRUD reminders; compute overdue
- Mark done/dismissed; trigger notifications

Collaborators:

- ReminderRepository, NotificationService, PatientService, AuditService

---

18. AttachmentService

Responsibilities:

- Upload/download via signed URLs
- Enforce "visit XOR exam" link constraint (or patient-level)
- Virus/type checks; manage file lifecycle

Collaborators:

- AttachmentRepository, ObjectStorage, VisitService, ExamService, PatientService, AuditService

---

19. TemplateService

Responsibilities:

- Create new template versions
- Validate field schemas
- Deactivate templates; ensure immutability of versions

Collaborators:

- ExamTemplateRepository, AuditService

---

20. ExamService

Responsibilities:

- Create/update exams; validate results_json against template version
- Finalize exam (prevent further edits)
- Manage exam attachments; ensure visit exists

Collaborators:

- ExamRepository, ExamTemplateRepository, VisitService, AttachmentService, TemplateService, AuditService

---

21. SearchService

Responsibilities:

- Full-text / filtered search (patients, owners)
- Paging/sorting results
- Support for visit and medication search (future)

Collaborators:

- PatientRepository, OwnerRepository, VisitRepository, DB indexes

---

22. ExportService

Responsibilities:

- CSV/JSON exports of patient data
- Backup triggers and data archival

Collaborators:

- All Repositories, ObjectStorage, AuditService

---

23. AuditService

Responsibilities:

- Write AuditLog entries
- Retrieve audit trails per entity/user
- Support compliance queries

Collaborators:

- AuditRepository, User

---

24. NotificationService

Responsibilities:

- Send reminder emails/SMS
- Templates for messages
- Queue notification delivery

Collaborators:

- EmailGateway, SmsGateway, ReminderService, User

C) Infrastructure (Gateways & Repositories)

25. ObjectStorage

Responsibilities:

- Sign upload/download URLs for secure file access
- Delete objects; manage file lifecycle in S3

Collaborators:

- AttachmentService

---

26. Repositories

Responsibilities:

- Persistence for each entity (CRUD, queries)
- Database access abstraction

Classes:

- UserRepository, OwnerRepository, PatientRepository, VisitRepository
- MedicationRepository, DoseEventRepository, ReminderRepository
- AttachmentRepository, ExamTemplateRepository, ExamRepository, AuditRepository

Collaborators:

- Respective Services + Entities

---

27. EmailGateway

Responsibilities:

- Abstract external email providers
- Send emails with templates

Collaborators:

- AuthService, NotificationService

---

28. SmsGateway

Responsibilities:

- Abstract external SMS providers
- Send SMS notifications

Collaborators:

- NotificationService

Class Diagram

These are partial diagrams, due to the high number of classes a high definition diagram is uploaded in the repository.

Diagram 1: Domain Entities and Relationships

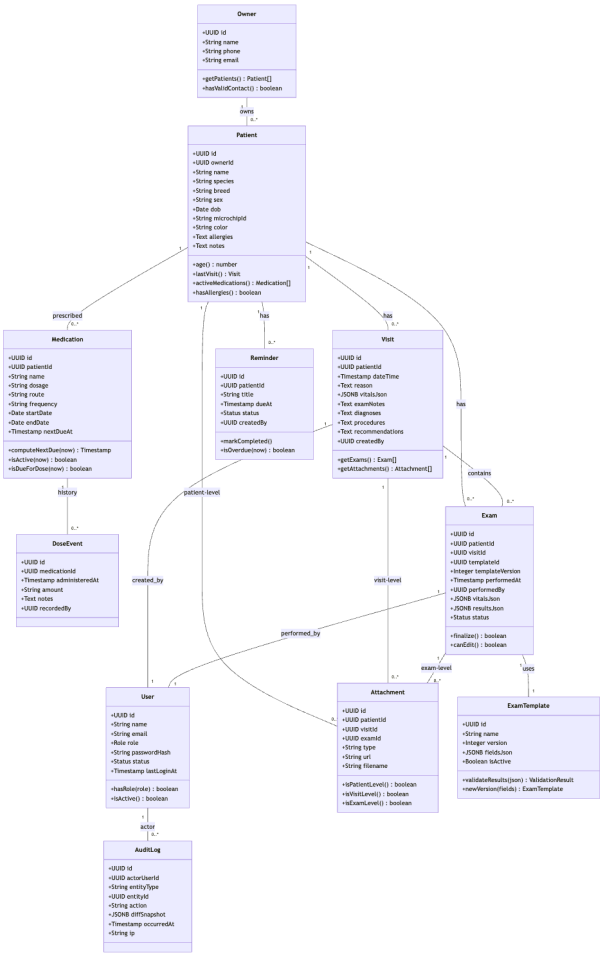
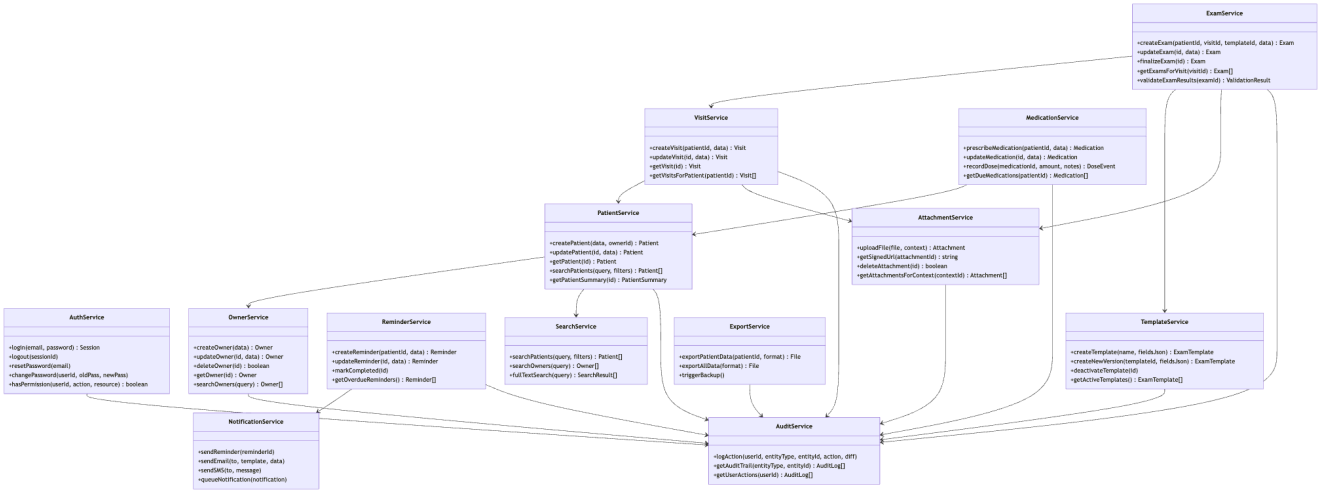


Diagram 2: Application Services Layer



Business Model Processes

Process: Register a New Patient Visit

Role in the app

This is the heart of VetTrack: it gets a patient into the system with the right context (patient, visit details, optional exams/attachments, and follow-ups). It’s what enables the summary view, timelines, reminders, and treatment tracking to stay current.

Starting a Visit

When a patient arrives at the clinic, the veterinarian begins by searching for the patient record using the animal's name, microchip number, or owner information. If the patient already exists in the system, the vet can immediately proceed to create a new visit. If this is a first-time patient, the system prompts the veterinarian to create the owner's contact information first, then the patient record, ensuring all necessary information is captured before the consultation begins.

Documenting the Consultation

Once the patient profile is open, the veterinarian creates a new visit entry and records the essential consultation details: the reason for the visit, clinical observations, vital signs, diagnoses, procedures performed, and any recommendations for the owner. This structured approach ensures that all relevant information is captured in a consistent format, making it easy to reference during future visits.

Concurrent Clinical Tasks

After the core visit details are entered, the veterinarian can perform several tasks in parallel, depending on the needs of the consultation. Files such as X-rays, lab results, or videos can be uploaded and automatically stored in secure object storage, with the system linking these files to the visit record. At the same time, the vet can prescribe new medications or update existing treatment plans, and create reminders for follow-up appointments or upcoming vaccinations. This parallel workflow reflects the reality of clinical practice, where multiple aspects of patient care are addressed during a single visit.

Structured Examinations

For certain types of consultations, such as surgical procedures, diagnostic tests, or specialized evaluations, the veterinarian can create a structured exam based on a predefined template. The system guides the vet through the required fields, captures specific vitals and results, and validates the entered data against the template schema. This ensures that important clinical information is consistently recorded and can be reliably compared across multiple examinations of the same type.

System Automation and Audit Trail

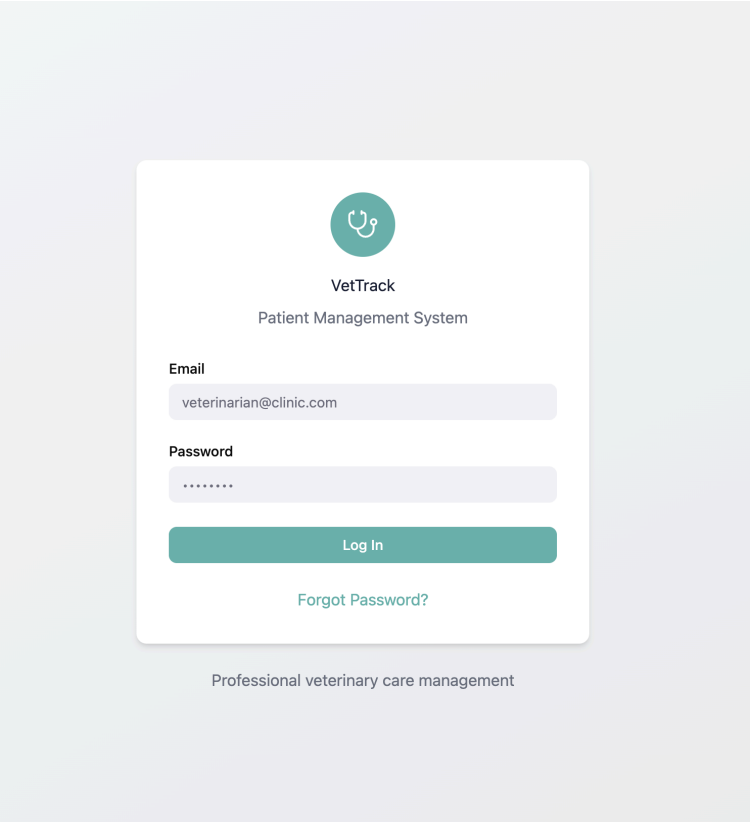
Throughout the process, VetTrack operates quietly in the background, handling validation, file storage, and record-keeping. Once the veterinarian completes the visit documentation, the system writes comprehensive audit log entries that track what was created or updated, who performed the actions, and when they occurred. This automatic auditing supports both clinical accountability and troubleshooting, without requiring any additional effort from the veterinarian.

References (BPMN basics)

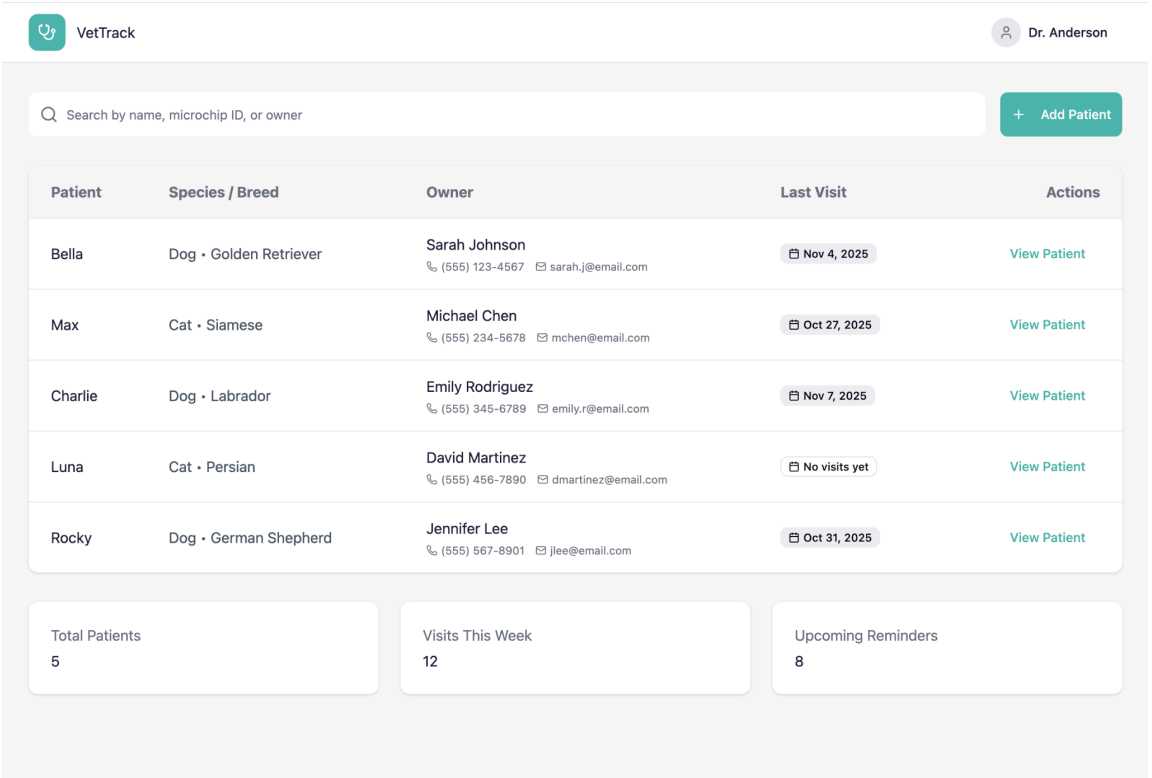
- What BPMN is and why it's used (standard notation across stakeholders). Visual Paradigm
- Flow elements (Activities/Events/Gateways) & connecting objects (Sequence/Message flows). Visual Paradigm
- Pools & Lanes (participants/roles). Visual Paradigm
- Gateway types (Exclusive, Parallel). Camunda
- Data objects/stores and notation overview. Visual Paradigm

Mockups/Wireframe

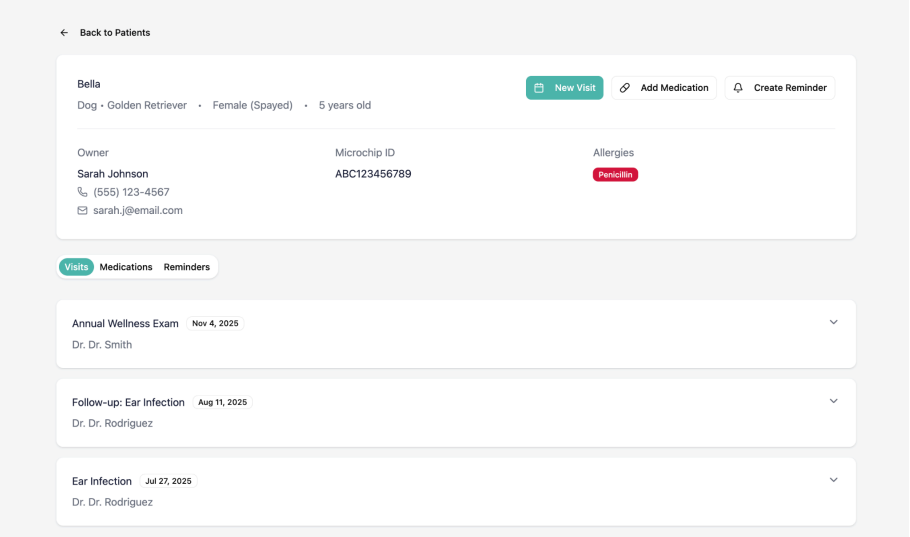
1. Login Page: Entry point for the whole application, here the users will log into their account to access their patients.

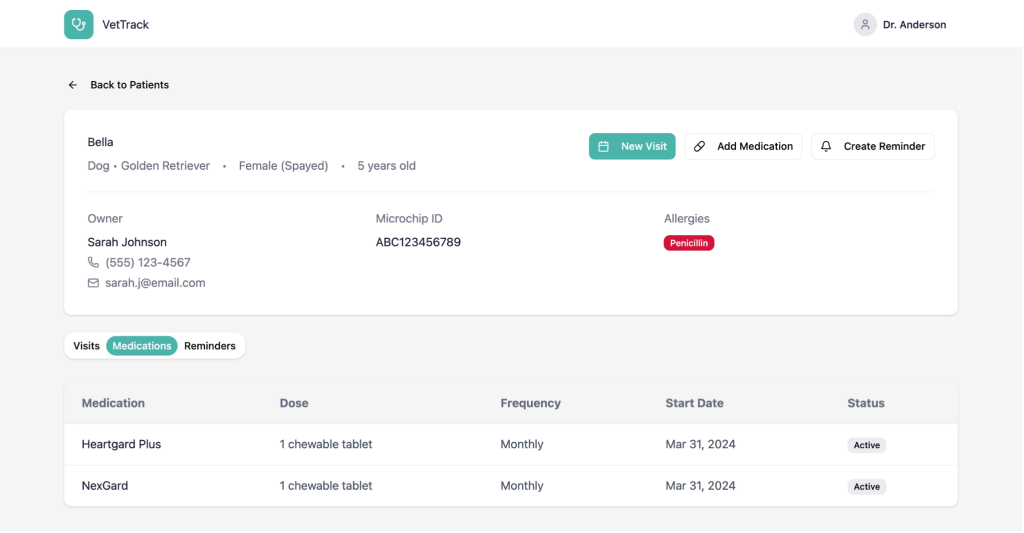


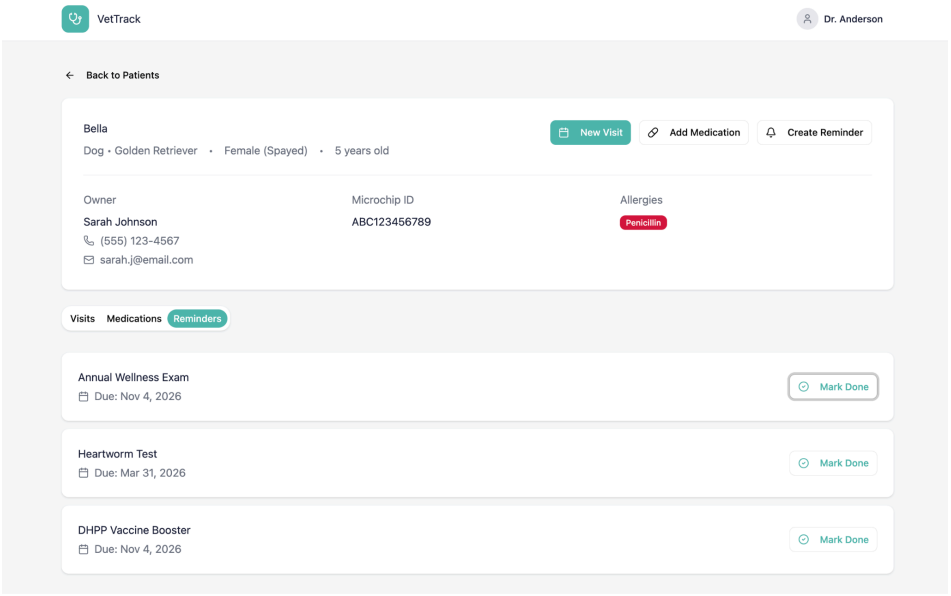
2. Main page: Here the user can see their current patients, search for a particular patient, add another patient and get a quick summary of their work for the week.



3. Patient Detail: This view provides a summary of the patient details, an option to add new visits, register medications and create custom reminders. We can see a chronological list for the visits, current medications and reminders for the patient.







4. Add Patient Form: Here we are able to register a new patient for the user, including their owner information.