MediDelivery — Detailed Project Report

Generated: February 23, 2026

Executive Summary

MediDelivery is a full-stack Django web application built as a college-level project to demonstrate an online medicine ordering, payment, and delivery workflow. The application provides:

- Medicine browsing with images and details
- A session-based shopping cart with quantity controls
- Checkout supporting Cash-on-Delivery (COD) and Razorpay payment gateway
- Order tracking for customers
- An admin portal to view orders and assign delivery personnel
- A delivery portal for delivery staff to update order status (strict progression)

The codebase is organized into three primary Django apps: `core`, `adminapp`, and `delivery`, and uses SQLite for storage in development.

Contents

Project Goals

- Build an intuitive interface for customers to order medicines.
- Integrate a payment gateway (Razorpay) to demonstrate payment flows.
- Implement role-based functionality: admin and delivery users.
- Keep the project small, modular, and easily runnable for demonstration.

Technologies

- Python 3.x + Django
- SQLite (development)
- Django templates and CSS (`core/static/core/css/style.css`)

- Razorpay checkout for payments
- Mermaid diagrams embedded in this Markdown for visuals (requires a renderer that supports Mermaid)

Architecture Overview

- `config/` — project settings and URL routing (`config/settings.py`, `config/urls.py`). Environment variables loaded via `.env`.
- `core/` — main app implementing product catalog, cart, checkout, payments, orders, profiles, and user auth.
- `adminapp/` — admin-facing dashboard and order assignment flows.
- `delivery/` — delivery user login, assigned order listing, and status update flows.
- `media/` — uploaded images (medicine photos).

Data Models (detailed)

Below is a concise description of the core data models and important fields. For full model code, see `core/models.py`.

- `Medicine`
  - `id` (auto PK), `name`, `brand`, `price`, `stock`, `description`, `image` (upload to `medicines/`), `is_active`
- `User` (built-in `django.contrib.auth.models.User`)
- `Profile`
  - `user` (OneToOne -> `User`), `role` (e.g., `admin`, `delivery`, default customer), `phone`, `address` (used in decorators and forms)
- `Order`
  - `id` (PK), `user` (FK -> `User`), `full_name`, `phone`, `address`, `status` (choices: `placed`, `packed`, `shipped`, `delivered`, `cancelled`), `payment_method` (`cod` | `razorpay`), `is_paid` (bool), `created_at`, `assigned_delivery` (nullable FK -> `User` for delivery staff), `razorpay_order_id` (optional, set when creating Razorpay order)
- `OrderItem`
  - `order` (FK -> `Order`), `medicine` (FK -> `Medicine`), `quantity`, `unit_price`

ER Diagram (Mermaid)

```
erDiagram
  USER ||--o{ PROFILE : has
  USER ||--o{ ORDER : places
  ORDER ||--o{ ORDERITEM : contains
  MEDICINE ||--o{ ORDERITEM : "is referenced by"
  USER ||--o{ ORDER : "assigned_delivery (nullable)"

  USER {
    integer id PK
    string username
    string email
  }
  PROFILE {
    integer id PK
    integer user_id FK
    string role
    string phone
    text address
  }
  MEDICINE {
    integer id PK
    string name
    string brand
    decimal price
    integer stock
  }
  ORDER {
    integer id PK
    integer user_id FK
    integer assigned_delivery_id FK nullable
    string status
    string payment_method
    boolean is_paid
    datetime created_at
  }
  ORDERITEM {
    integer id PK
    integer order_id FK
    integer medicine_id FK
    integer quantity
    decimal unit_price
  }
```

Working Flow (user journey)

This describes the typical sequence from browsing to delivery completion.

```
flowchart LR
  A[User: Browse Medicines] --> B[View Medicine Detail]
  B --> C[Add to Cart]
  C --> D[Cart View]
  D --> E[Checkout]
  E --> F{Payment Method}
  F -->|COD| G[Create Order (is_paid: false)]
  F -->|Razorpay| H[Start Razorpay Checkout]
  H --> I[Payment Success]
  I --> J[Mark Order is_paid = true]
  J --> G
  G --> K[Order Placed]
  K --> L[Admin assigns delivery]
  L --> M[Delivery user receives assignment]
  M --> N[Delivery updates status: packed -> shipped -> delivered]
  N --> O[Order Completed]
```

Notes:

- The application enforces strict delivery-status progression in `delivery/views.py` (`placed` → `packed` → `shipped` → `delivered`).
- For Razorpay payments, the deployment must verify webhook/signatures for security; the current checkout uses client-side integration plus a `razorpay_callback` endpoint.

Function / Request Flow (views and handlers)

This diagram shows main endpoints and how they relate.

```
flowchart TD
  subgraph Core
    H1(Home): / -> `core.views.home`
    H2(Medicine Detail): /medicine/<id>/ -> `core.views.medicine_detail`
    H3(Cart): /cart/ -> `core.views.cart_view`
    H4(Checkout): /checkout/ -> `core.views.checkout`
    H5(Start Razorpay): /pay/<order_id>/ -> `core.views.start_razorpay_payment`
    H6(Razorpay Callback): /razorpay/callback/ ->
`core.views.razorpay_callback`
    H7(Order Detail): /my-orders/<order_id>/ -> `core.views.order_detail`
  end

  subgraph Admin
    A1(Admin Dashboard): /admin-panel/admin-dashboard/ ->
`adminapp.views.admin_dashboard`
    A2(Assign): /admin-panel/assign-delivery/<order_id>/ ->
`adminapp.views.assign_delivery`
  end

  subgraph Delivery
    D1(Delivery Login): /delivery/login/ -> `delivery.views.delivery_login`
    D2(Delivery Dashboard): /delivery/ -> `delivery.views.delivery_dashboard`
    D3(Update Status): /delivery/order/<order_id>/status/ ->
`delivery.views.delivery_update_status`
  end

  H1 --> H2 --> H3 --> H4
  H4 -->|razorpay| H5 --> H6 --> H7
  H4 -->|cod| H7
  H7 --> A1
  A1 --> A2 --> D2
  D2 --> D3
```

Key Files & Responsibilities

- `config/settings.py` — loads `.env` (Razorpay keys) and configures installed apps.
- `config/urls.py` — routes `core`, `delivery`, and `adminapp` apps and serves static/media in DEBUG.
- `core/models.py` — `Medicine`, `Order`, `OrderItem`, `Profile` and post-save user.Profile signal.
- `core/views.py` — cart helpers (`_get_cart`, `_save_cart`), cart endpoints, checkout, `start_razorpay_payment`, `razorpay_callback`, auth views, profile, and order views.
- `core/templates/core/` — templates for `home.html`, `medicine_detail.html`, `cart.html`, `checkout.html`, `order_detail.html`, `order_success.html`, `payment_failed.html`, and `razorpay_checkout.html`.
- `core/static/core/css/style.css` — primary styles.
- `adminapp/views.py` — `admin_dashboard`, `assign_delivery`, and `admin_login`; uses `adminapp/utils.py` decorator for role enforcement.
- `delivery/views.py` — `delivery_dashboard`, `delivery_order_detail`, `delivery_update_status`, protected via `delivery/utils.py`.
- `.env` — contains `RAZORPAY_KEY_ID` and `RAZORPAY_KEY_SECRET` (example keys present for testing).

Setup & Run (development)

1. Create & activate a virtual environment (PowerShell example):

```
python -m venv .venv
.venv\Scripts\Activate.ps1
```

2. Install dependencies:

```
pip install -r requirements.txt
```

3. Ensure `.env` is present at project root with Razorpay keys (sample file already in workspace).

4. Apply migrations and create a superuser:

```
python manage.py migrate
python manage.py createsuperuser
```

5. (Optional) collect static, then run the dev server:

```
python manage.py collectstatic --noinput
python manage.py runserver
```

6. Visit:

- Application: `http://127.0.0.1:8000/`
- Admin: `http://127.0.0.1:8000/admin/`
- Admin portal: `http://127.0.0.1:8000/admin-panel/admin-dashboard/`
- Delivery portal: `http://127.0.0.1:8000/delivery/`

Exporting this report

- This Markdown contains Mermaid blocks. To render diagrams you need a Markdown viewer that supports Mermaid (VS Code + Mermaid Preview, GitHub, GitLab, or certain static site generators).
- To convert to PDF: open in a renderer and print to PDF (Ctrl+P).
- To convert to DOCX with `pandoc` (Mermaid may not render directly):
  - Option A: Render mermaid diagrams to images (using `mmdc` / `mermaid-cli`) and replace code blocks with image includes, then run `pandoc`.
  - Option B: Convert the HTML version to DOCX using `pandoc` after ensuring diagrams are rendered.

Example `pandoc` command (may require extra handling for Mermaid):

```
pandoc MediDelivery_Detailed_Report.md -o MediDelivery_Detailed_Report.docx
```

Security & Production Recommendations

- Move `SECRET_KEY` and all secrets to environment variables and remove hardcoded secrets from `config/settings.py`.
- Set `DEBUG = False` and configure `ALLOWED_HOSTS` for production.
- Verify Razorpay payment signatures on callbacks and/or use server-side webhooks with signature verification.
- Replace SQLite with PostgreSQL (or other production-grade DB).
- Serve static files and media from a dedicated storage (S3 + CDN or similar) and front with Nginx.
- Add logging for payments and admin/delivery actions.

Testing & Validation

- Create unit tests for:
  - Cart helpers: add/inc/dec/remove/update
  - Checkout: order creation, stock decrement, payment flow
  - Role decorators: `adminapp.utils.admin_required`, `delivery.utils.delivery_required`
  - Delivery status enforcement: `delivery_update_status` progression rules

- Manual testing checklist:
    - Place a COD order and verify `Order.is_paid` remains false and the order appears in admin dashboard.
    - Start a Razorpay payment (test keys in `.env`) and simulate success/failure flows.
    - Assign an order to a delivery user from the admin portal and verify assignment propagation to delivery dashboard.

Improvements & Future Work

- Add email notifications (order placed, assigned, delivered).
- Add search/indexing and filters for medicines (by category, generic name).
- Add pagination and caching for the medicines list.
- Add a REST API (Django REST Framework) for mobile clients.
- Implement retry logic and robust logging for payment failures.

Appendix — Important file locations

- `config/settings.py` — configuration and environment loading
- `config/urls.py` — top-level URL includes
- `core/models.py`, `core/views.py`, `core/admin.py`, `core/templates/core/`
- `delivery/views.py`, `delivery/utils.py`, `delivery/templates/delivery/`
- `adminapp/views.py`, `adminapp/utils.py`, `adminapp/templates/admin/`
- `report/` — this report files (`MediDelivery_Detailed_Report.md`, `MediDelivery_Report.md`, `MediDelivery_Report.html`)

If you'd like, I can now:

- Render Mermaid diagrams to PNG and embed them in this Markdown,
- Convert this Markdown to PDF here (I may need headless Chrome/wkhtmltopdf), or
- Produce a DOCX using `pandoc` (if you want me to attempt and your system has `pandoc`).

Which export or next step do you want? If you want a PDF now, I can attempt to produce one and save it in `report/`.