Savannah Informatics Backend Engineer Assessment

Author: Daniel Muuo Muumbi

Live Demo:

• Admin Interface: https://savannah-informatics-assignment.onrender.com/admin/ (username: root, password: savannah2025)

• Customer Portal: https://savannah-informatics-assignment-3.onrender.com/

Source Code: https://github.com/Danmuumbi/savannah_informatics

1. Introduction

This project was built as part of the Savannah Informatics Backend Engineer technical interview. It demonstrates the design and deployment of a full-stack backend service using Python and Django, while showcasing:

- RESTful API design
- Secure authentication (OAuth2/OpenID Connect)
- Relational database design with hierarchical product categories
- Automated notifications (SMS & Email)
- Containerization & cloud deployment (Docker + Render)

The aim was to create a production-ready prototype that is both scalable and easy to understand, for recruiters and engineers alike.

Why the Application is Divided into Frontend and Backend

This solution is intentionally organized into two distinct parts:

1. Frontend (Customer Portal)
https://savannah-informatics-assignment-3.onrender.com/
This is the public-facing side of the system, created to showcase my ability to implement modern authentication using OAuth2 / OpenID Connect.
Here, customers can:

• Sign in securely using Google sign-in or receive a one-time login link by email.
• After successful login, view their own orders in a simple, user-friendly dashboard.

2. Backend (Admin Dashboard)
https://savannah-informatics-assignment.onrender.com/admin/
This is the administrative side of the system, where the core business logic resides.
Here, the admin can:

• Manage users, customers, products, categories, and orders.
• Trigger automated notifications:
 - SMS alerts to customers using the Africa's Talking SMS gateway.
 - Email notifications to the administrator with order details.

2. Key Features at a Glance

- Admin Dashboard – Manage customers, users, categories, products, and orders through a secure Django admin panel. (login)
- Role Management – Use of Django Groups to assign different privileges to team members.
- Customer Portal – Simple, user-friendly login via Google sign-in (OAuth2/OpenID) or email one-time login link. (login after adding customer)
- Order Notifications – SMS alert to the customer using Africa's Talking sandbox and email notification to the administrator.
- Hierarchical Categories – Products can belong to categories of any depth (e.g. Bakery → Bread → Wholemeal).
- REST API Endpoints – checking customer existence and retrieving their orders.
- Deployment – Dockerized and hosted on Render for easy scalability.

3. System Overview

Architecture:
[Customer Frontend] -> [Django Backend] -> [PostgreSQL DB]

Backend: Python 3 + Django + Django REST Framework
Database: PostgreSQL
Containerization: Docker
CI/CD: GitHub + Render deployment
Messaging: Africa's Talking SMS sandbox & Email SMTP

4. How It Works

Admin Side:
1. Login: Use the admin credentials to access the Django admin panel.
2. Manage Users: Add co-workers and assign them to groups with specific privileges.
3. Customers: Create customer records with name, email, phone, and address.
4. Categories & Products: Build multi-level product categories and add products under them.

5. Orders: Place orders on behalf of customers. Triggers an SMS to the customer and an email to the administrator instantly.

Customer Side:
1. Pre-registration: The admin must add the customer first.
2. Login Options: Continue with Google (OAuth2/OpenID) or Manual Email Login – Customer enters name & email and receives a one-time secure login link.
3. Dashboard: Customers can view all their past orders.

5. REST API Examples

Sample endpoints powering the frontend:
- /api/check_customer (GET): Verify if a customer exists by email & name.
- /api/customer_orders (GET): Retrieve all orders belonging to a given customer.

Responses are JSON formatted, making it easy for the frontend to consume.

6. Testing

Unit Tests: Implemented for key admin operations such as adding customers and managing orders.
Coverage: Basic coverage achieved; due to time constraints full e2e testing was not completed.

7. Deployment

Containerization: The entire project is Dockerized for portability.
Hosting: Deployed to Render using Docker images.
Version Control & CI/CD: GitHub repository with clear commit history and automatic deployment triggered by Git pushes.

8. Getting Started Locally (For Developers)

For technical reviewers who wish to run it locally:

```
git clone https://github.com/Danmuumbi/savannah_informatics.git
cd savannah_informatics
docker-compose up --build
```

The app will be available at http://localhost:8000/.

9. Future Improvements

- Add full integration & end-to-end tests for higher coverage.
- Enhance the frontend with richer UI/UX features.

- Expand SMS & email notifications for more event types.
- Kubernetes deployment for large-scale production environments.

10. Acknowledgements

Special thanks to Savannah Informatics for providing this engaging and challenging assessment. This project reflects my commitment to clean code, secure practices, and scalable architecture.

Daniel Muuo Muumbi
Software Engineer
 muuomuubi@gmail.com | +254 714 202 946
GitHub | LinkedIn | Portfolio