INTERNSHIP REPORT

Company: Ririana Innovations Private Limited

Submitted by: Siddareddy Rajesh Reddy

Duration: 05/12/2025 - 13/02/2026

Department: System Developer

SUMMARY OF MY INTERN WORK

This report presents a comprehensive account of the work undertaken during my internship at Ririana Innovations Private Limited. The internship focused on full-stack web development using Django framework, encompassing database design, RESTful API development, authentication and authorization systems, algorithm implementation, payment gateway integration, and cloud deployment.

The project involved building a complete backend system with PostgreSQL database integration, JWT-based authentication, and Razorpay payment processing capabilities for a event app. The application's database was initially deployed on Render cloud platform for hosting and testing. Due to the expiration of Render's free trial period, a strategic database migration was undertaken, transitioning from Render's managed PostgreSQL to Neon's serverless PostgreSQL platform. This migration demonstrated adaptability to changing infrastructure requirements while maintaining application stability and data integrity.

Key achievements include the successful development of secure authentication mechanisms, implementation of role-based authorization, integration of payment processing capabilities, and hands-on experience with cloud deployment and database migration strategies.

TABLE OF CONTENTS

1. Introduction
2. Project Setup and Database Configuration
3. RESTful API Development
4. Authentication and Authorization Implementation
5. Algorithm Design
6. Application Backend Development
7. Payment Gateway Integration
8. Key Learnings and Skills Acquired
9. Challenges and Solutions
10. Conclusion

1. INTRODUCTION

1.1 Internship Objectives

The primary objectives of this internship were to: - Gain hands-on experience in Django web framework development - Understand database design and integration with PostgreSQL - Implement secure authentication and authorization mechanisms - Develop and test algorithms for real-world applications - Integrate third-party payment gateway services - Build production-ready backend systems

1.2 Project Scope

The internship project involved developing a complete Django-based backend system with the following components: - PostgreSQL database integration - CRUD operations via RESTful APIs - JWT-based authentication system - Authorization framework implementation - Custom algorithm development and testing - Razorpay payment gateway integration - Cloud deployment on Render platform - Database migration to Neon serverless PostgreSQL - Production environment configuration

---

2. PROJECT SETUP AND DATABASE CONFIGURATION

2.1 Django Project Initialization

The first phase of the internship involved setting up the development environment and creating a new Django project. This included:

Activities Undertaken: - Installation of Python 3.8+ and Django 4.2.7 - Creation of virtual environment for dependency isolation - Project structure initialization using Django's startproject command - Configuration of project settings and middleware

Key Configuration:

```
- Framework: Django 4.2.7
- Python Version: 3.8+
- Project Structure: Modular app-based architecture
```

2.2 PostgreSQL Database Integration

PostgreSQL was selected as the primary database management system due to its robustness, scalability, and advanced features.

Initial Setup - Local Development: 1. Installation and configuration of PostgreSQL server locally using PgAdmin4 2. Creation of dedicated database for the project 3. Configuration of database connection parameters in Django settings 4. Installation of psycopg2-binary adapter for PostgreSQL-Django communication 5. Testing database connectivity and query execution

Database Configuration: - Database Engine: PostgreSQL 14+ - Connection pooling enabled for performance optimization - Secure credential management using environment variables - Database timezone configuration for accurate timestamp handling - Query optimization settings for improved performance

Initial Deployment Database - Render: During the initial deployment phase, the application was configured to use Render's managed PostgreSQL database service: - Render PostgreSQL instance provisioned - Connection string configured in environment variables

2.3 Database Migration to Neon

Migration Background: Following the expiration of Render's free trial period, a immidiate decision was made to migrate the database to Neon, a serverless PostgreSQL platform offering a more sustainable free tier for continued development and testing.

Migration Process:

Step 1: Neon Account Setup and Database Provisioning - Created account on Neon platform (https://neon.tech) - Provisioned new PostgreSQL database instance - Configured database region for optimal latency - Obtained connection string and credentials - Enabled connection pooling for efficient resource utilization

Step 2: Data Backup from local database - Created complete database backup using pg_dump from PgAdmin4 - Exported database schema and data - Verified backup integrity and completeness - Documented current database state and row counts - Backed up environment variables and configuration

Step 3: Data Migration to Neon - Verified data integrity post-migration - Checked all table structures and relationships - Validated data consistency across all models - As It was test data, no data was recovered from render

Step 4: Application Configuration Update - Updated DATABASE_URL environment variable with Neon connection string - Modified Django settings for Neon-specific configurations

Step 5: Testing and Validation - Ran Django migrations to ensure schema consistency - Checked all operations through frontend and backend - Verified all CRUD operations functionality - Tested authentication and authorization flows

2.4 Database Schema Design

A comprehensive database schema was designed to support the application requirements, including: - User management tables - Chat Feature tables - Admin related table - Job relatedt tables - Notification related Tables

---

3. RESTFUL API DEVELOPMENT

3.1 API Architecture

Django REST Framework (DRF) was utilized to build a comprehensive set of RESTful APIs supporting all CRUD (Create, Read, Update, Delete) operations.

Framework Used: - Django REST Framework (DRF) - Serializers for data validation and transformation - ViewSets for organized endpoint management - URL routing with RESTful conventions

3.2 CRUD Operations Implementation

Complete CRUD functionality was implemented with the following endpoints:

Create Operations: - POST endpoints for resource creation - Request validation using DRF serializers - Proper HTTP status code responses (201 Created)

Read Operations: - GET endpoints for individual resource retrieval - List endpoints with pagination support - Filtering and search capabilities

Update Operations: - PUT endpoints for complete resource updates - PATCH endpoints for partial updates - Optimistic locking to prevent race conditions

Delete Operations: - DELETE endpoints with soft delete capability - Cascade deletion handling for related records - Confirmation mechanisms for destructive operations

3.3 API Testing with Postman

Comprehensive API testing was conducted using Postman to ensure reliability and correctness.

Testing Activities: - Created Postman collection for all endpoints - Validated request/response payloads - Tested error handling and edge cases - Verified HTTP status codes and response formats —

4. AUTHENTICATION AND AUTHORIZATION IMPLEMENTATION

4.1 Understanding Authentication vs. Authorization

Authentication is the process of verifying the identity of a user or system. It answers the question: "Who are you?"

Authorization is the process of determining what actions an authenticated user is permitted to perform. It answers the question: "What are you allowed to do?"

Key Differences:

Aspect	Authentication	Authorization
Purpose	Verify identity	Verify permissions
Process	Login credentials	Access control checks
Timing	First step	After authentication
Example	Username/password	User role permissions

4.2 JWT Token-Based Authentication

A custom user authentication system was implemented using JSON Web Tokens (JWT) for stateless, secure authentication.

Implementation Components:

User Registration: - Custom user model creation - Password hashing using Django's built-in security features - Email validation and uniqueness constraints - User profile creation with additional metadata

Login API: - Credential verification against database - JWT token generation upon successful authentication - Access token and refresh token mechanism - Token expiration configuration (access: 15 minutes, refresh: 7 days)

Logout API: - Token invalidation mechanism - Blacklisting of refresh tokens - Session cleanup procedures

Token Management: - Secure token storage recommendations - Token refresh endpoint for seamless user experience - Automatic token validation middleware

4.3 Authorization Framework

An authorization system was developed to control access to resources based on user roles and permissions.

Authorization Example Implementation:

Role-Based Access Control (RBAC): - Definition of user roles (Admin, Manager, User, Guest) - Permission assignment to roles - Resource-level access control

Permission Checking: - Custom permission classes - Decorator-based permission enforcement - Object-level permissions for fine-grained control

---

5. ALGORITHM DESIGN

5.1 Algorithm Development

A custom algorithm was designed to support core application functionality. The algorithm development process included:

Design Phase: - Problem analysis and requirement gathering for job recommendation algorithm - Algorithm conceptualization and flowchart creation of k-means clustering - Complexity analysis (time and space) - Edge case identification

Documentation: - Detailed algorithm documentation - Input/output specifications - Pseudocode representation - Performance considerations

5.2 Django Integration

The algorithm was integrated into the Django application and thoroughly tested.

Integration Steps: 1. Implementation as Django utility functions 2. Creation of dedicated API endpoints for algorithm execution 3. Integration with database models for data persistence 4. Error handling and logging implementation

---

6. APPLICATION BACKEND DEVELOPMENT

6.1 Backend Architecture

A comprehensive backend system was developed following Django best practices and modular design principles.

Architecture Components: - Model layer: Database schema and business logic - View layer: Request handling and response generation - Serializer layer: Data validation and transformation - URL routing: Endpoint organization and versioning

6.2 Core Features Implementation

User Management: - Custom user model with extended profile information - Email and phone number validation - Password strength enforcement

Data Models: - Comprehensive model definitions for all entities - Relationship mapping (One-to-One, One-to-Many, Many-to-Many) - Database indexing for query optimization - Model validators and custom save methods

6.3 API Endpoints

A complete set of API endpoints was developed to support frontend integration:

Endpoint Categories: - User management endpoints (registration, profile, preferences) - Authentication endpoints (login, logout, token refresh) - Resource CRUD endpoints - Search and filtering endpoints - Reporting and analytics endpoints

---

7. PAYMENT GATEWAY

7.1 Razorpay Integration Overview

The final phase involved integrating Razorpay payment gateway to enable secure payment processing capabilities.

Technology Stack: - Payment Gateway: Razorpay - SDK: Razorpay Python SDK - Security: Signature verification for callback validation

7.2 Payment Features Implemented

Payment Order Creation: - API endpoint to create Razorpay orders - Amount calculation and currency handling - Customer information capture - Order ID generation and tracking

Payment Processing: - Integration with Razorpay checkout interface - Real-time payment status tracking - Secure payment callback handling - Payment signature verification for security

Payment Verification: - Cryptographic signature validation - Payment status update in database - Customer notification system - Receipt generation

Payment Status Management: - Multiple payment status tracking (created, authorized, captured, failed) - Payment history retrieval API - Status-based filtering and reporting - Real-time status updates

7.3 Refund Management System

Refund Capabilities: - Full refund processing - Partial refund support - Refund status tracking - Refund history maintenance

Refund API Endpoints: - Initiate refund endpoint with amount validation - Refund details retrieval - Refund status monitoring - Integration with Razorpay refund API

7.4 Database Schema for Payments

Payment Model Fields: - Order identifiers (Razorpay order ID, payment ID) - Payment signature for verification - Amount and currency information -

Payment status tracking - Customer details (name, email, contact) - Refund information and tracking - Timestamps for audit trail - Additional notes field

7.5 Security Implementation

Security Measures: - Environment variable management for API credentials - Signature verification for all payment callbacks - Secure API key handling - HTTPS enforcement for production - Input validation and sanitization - Preserved Api keys .ENV files

---

8. KEY LEARNINGS AND SKILLS ACQUIRED

8.1 Technical Skills

Django Framework: - Proficiency in Django - Experience with Django REST Framework - Middleware and signal implementation

Database Management: - PostgreSQL administration and optimization - Database schema design and normalization - Query optimization and indexing strategies - Transaction management and data integrity

API Development: - RESTful API design principles - API versioning and documentation - Error handling and status code management - API security best practices

Authentication & Authorization: - JWT token-based authentication - Role-based access control (RBAC) - Session management - Security best practices

Payment Integration: - Third-party API integration - Webhook handling and verification - Payment flow implementation - Security considerations for financial transactions

8.2 Professional Skills

- Version control using Git and Bit Bucket
- API testing and documentation with Postman
- Problem-solving and debugging techniques
- Code organization and best practices
- Time management and task prioritization
- Technical documentation writing
- Collaboration and communication in a professional environment

---

9. CHALLENGES AND SOLUTIONS

9.1 Database Integration Challenges

Challenge: Initial difficulties in configuring PostgreSQL connection with Django, particularly with authentication and connection parameters.

Solution: Thoroughly reviewed Django documentation for database configuration, used environment variables for secure credential management, and implemented connection pooling for improved performance.

9.2 JWT Token Management

Challenge: Understanding the lifecycle of JWT tokens and implementing proper token refresh mechanisms.

Solution: Studied JWT specifications and best practices, implemented both access and refresh tokens with appropriate expiration times, and created a token refresh endpoint for seamless user experience.

9.3 Payment Gateway Integration

Challenge: Understanding Razorpay's signature verification mechanism and implementing secure payment callbacks.

Solution: Carefully studied Razorpay documentation, implemented cryptographic verification using HMAC-SHA256, and thoroughly tested payment flows in test mode before production deployment.

9.4 Authorization Implementation

Challenge: Designing a flexible yet secure authorization system that could handle complex permission requirements.

Solution: Implemented role-based access control with custom permission classes, studied Django's built-in permission framework, and created reusable permission decorators for API endpoints.

---

10. CONCLUSION

10.1 Project Summary

The internship at Ririana Innovations Private Limited provided comprehensive exposure to modern web development practices using Django framework. The project successfully achieved all its objectives, resulting in a fully functional backend system with the following accomplishments:

- Complete Django project setup with PostgreSQL integration
- RESTful API implementation with full CRUD operations

- Secure JWT-based authentication system
- Robust authorization framework with role-based access control
- Custom algorithm development and integration
- Production-ready payment gateway integration with Razorpay

10.2 Current Status

Note: While the payment gateway integration and machine learning components have been successfully developed and tested independently, they have not yet been integrated into the main application. This modular approach allows for future integration when the application architecture is ready to support these advanced features.

10.3 Personal Growth

This internship has been instrumental in my professional development. The hands-on experience with industry-standard tools and practices has significantly enhanced my technical capabilities and provided valuable insights into software development workflows. The exposure to real-world challenges and problem-solving scenarios has prepared me for future roles in software engineering.

10.4 Acknowledgments

I would like to express my sincere gratitude to Ririana Innovations Private Limited for providing this valuable learning opportunity. Special thanks to my mentors Founder & Director, Ririana Innovations Private Limited, Srijana Bhattarai Ma'am and Our Product Manager at Ririana Innovations Private Limited, Mukesh Ramesh Thombare, team members Priyanshu Singh, who guided me throughout this internship and helped me grow both technically and professionally.

APPENDICES

Appendix A: Technology Stack Summary

- **Backend Framework:** Django 4.2.7
- **Database:** PostgreSQL
- **API Framework:** Django REST Framework
- **Authentication:** JWT (JSON Web Tokens)
- **Payment Gateway:** Razorpay Python SDK
- **Environment Management:** python-decouple
- **Testing Tools:** Postman, Django Test Framework

Appendix B: Key Learnings Summary

1. Full-stack web development with Django
2. Database design and PostgreSQL administration

3. RESTful API development and testing
4. Authentication and authorization implementation
5. Third-party API integration
6. Security best practices in web development
7. Professional software development workflows
8. Cloud Deployment