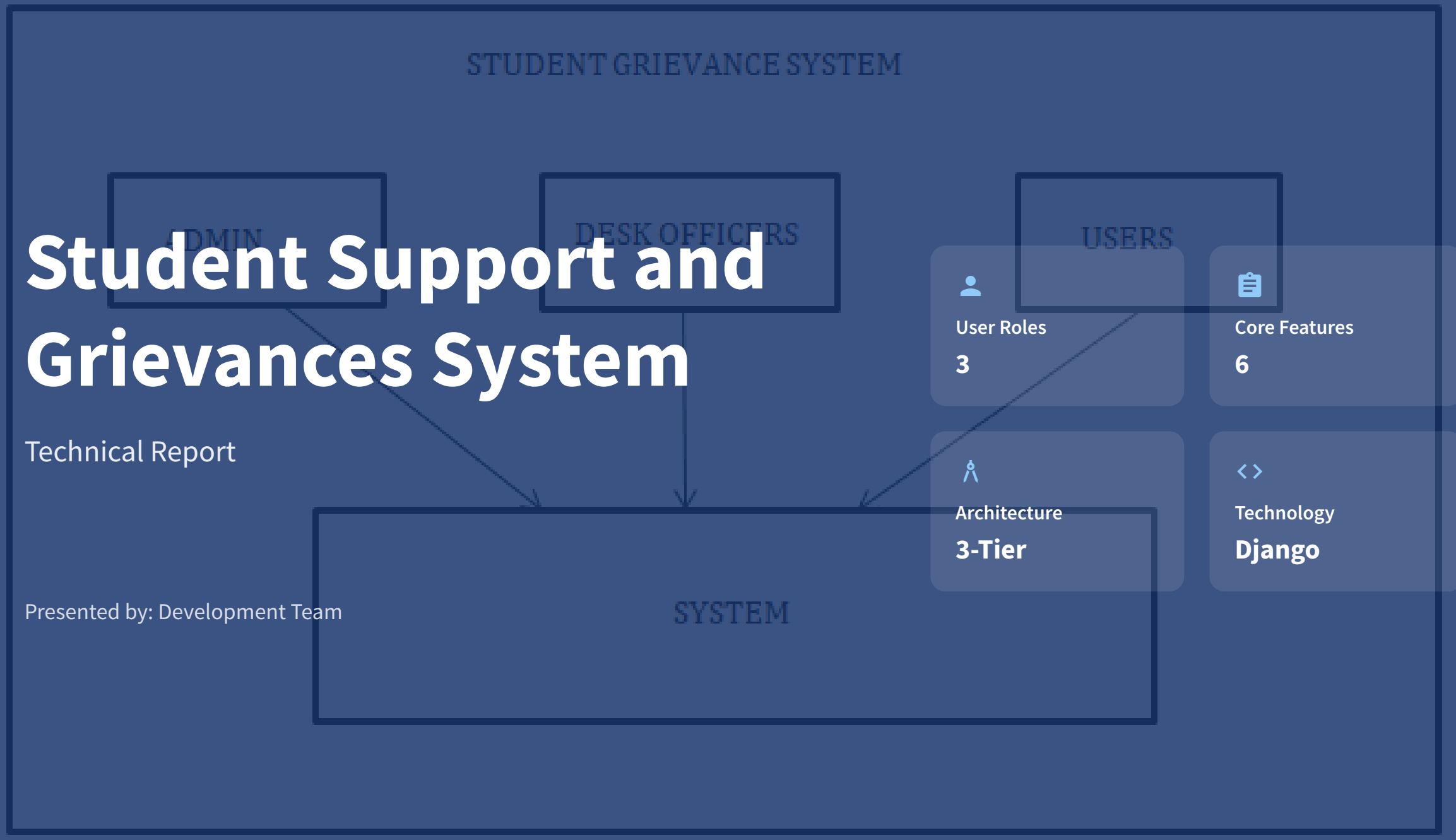



# Student Support and Grievances System

Technical Report

Presented by: Development Team

User Roles
3

Core Features
6

Architecture
3-Tier

Technology
Django

# Chapter 1: Introduction

## Project Title & Abstract

- **Web-based platform** for student grievances
- Replaces **traditional methods** (paper forms, emails)
- Improves **transparency & speed** of grievance handling

## Background & Problem

- No **centralized, trackable system** for grievances
- Leads to **mismanagement** of complaints
- Causes **slow responses** and student frustration

## Objectives

- Design **online platform** for grievances
- Implement **secure login** & profile management
- Enable **submission & tracking** of complaints
- Create **admin dashboards** & reporting

## Scope & Limitations

- **In scope:** Registration, submission, tracking, admin features
- **Out of scope:** Mobile app, voice/chat intake, external integrations
- **Limitations:** SQLite for prototype, institutional email dependency

# Chapter 2: Requirements

## Functional Requirements

- **User Management**: Registration & login
- **Submission**: Detailed grievances
- **Assignment**: Admin assigns to staff
- **Status Updates**: Track progress
- **Reporting**: Analytics & summaries

## Non-functional Requirements

- **Usability**: Intuitive interface
- **Performance**: <2s response time
- **Security**: HTTPS, hashed passwords
- **Compatibility**: Modern browsers
- **Scalability**: Future growth

## Use Cases / User Stories

- **Student**: Submit & track complaints
- **Admin**: Review & assign complaints
- **Staff**: Update status & resolution
- **Reports**: Analytics & insights

## Constraints

- **SQLite** for prototype (limits concurrency)
- **Email dependency** for notifications
- **Cloud deployment** (AWS/PythonAnywhere)

## Assumptions

- **Reliable internet** access on campus
- **Unique institutional emails** for users
- **SMTP service** available for alerts

# Chapter 3: Architecture

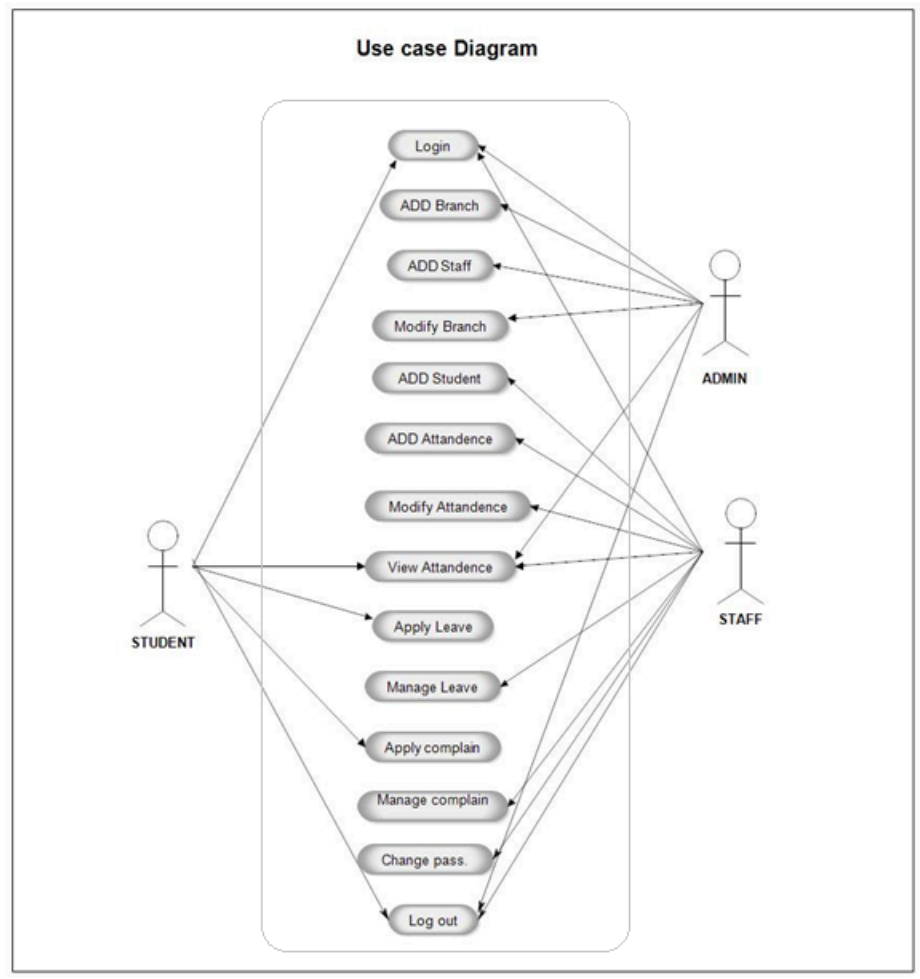
## 3-Tier Architecture

### Presentation Layer
HTML/CSS/JS (Bootstrap) frontend

### Application Server
Python/Django handles requests & logic

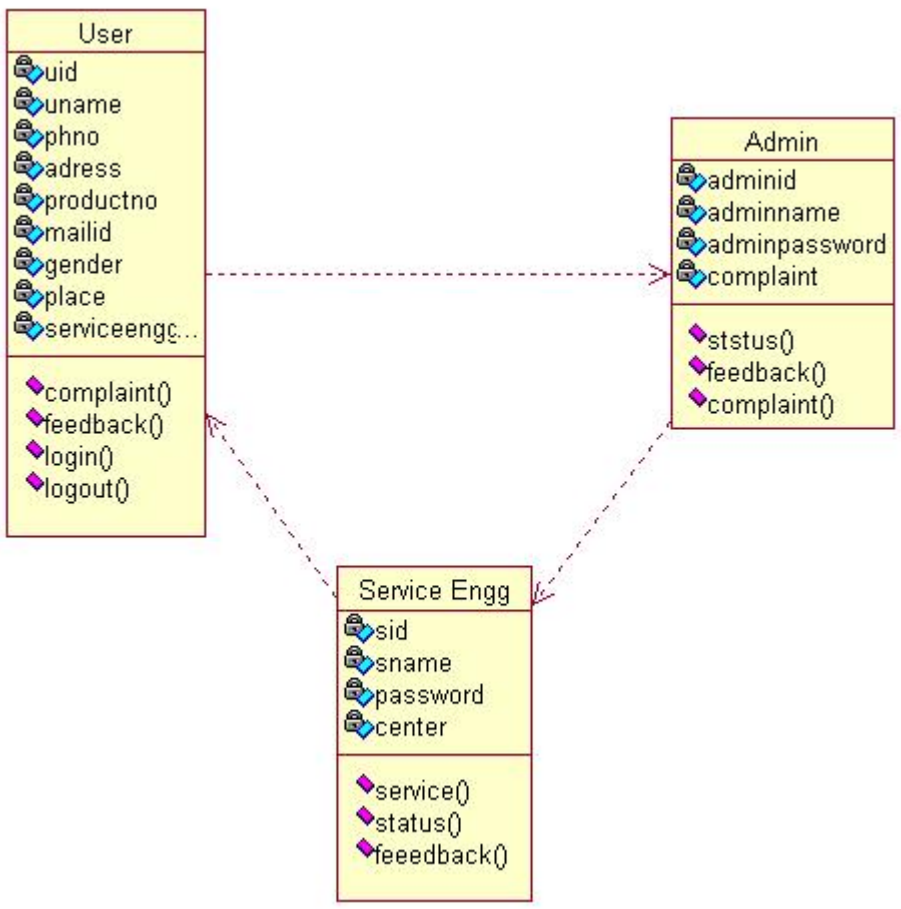
### Database Layer
SQLite (prototype) / PostgreSQL

## Use Case Diagram



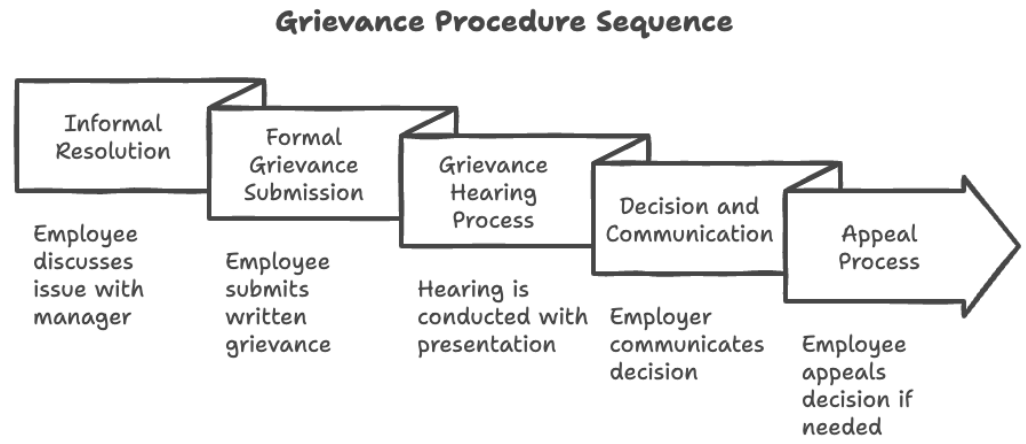

System interactions with Student, Admin, and Staff actors

## UML Class Diagram




- **Student**: ID, name, email
- **Grievance**: title, description, status
- **User**: Admin/Staff with role-based access

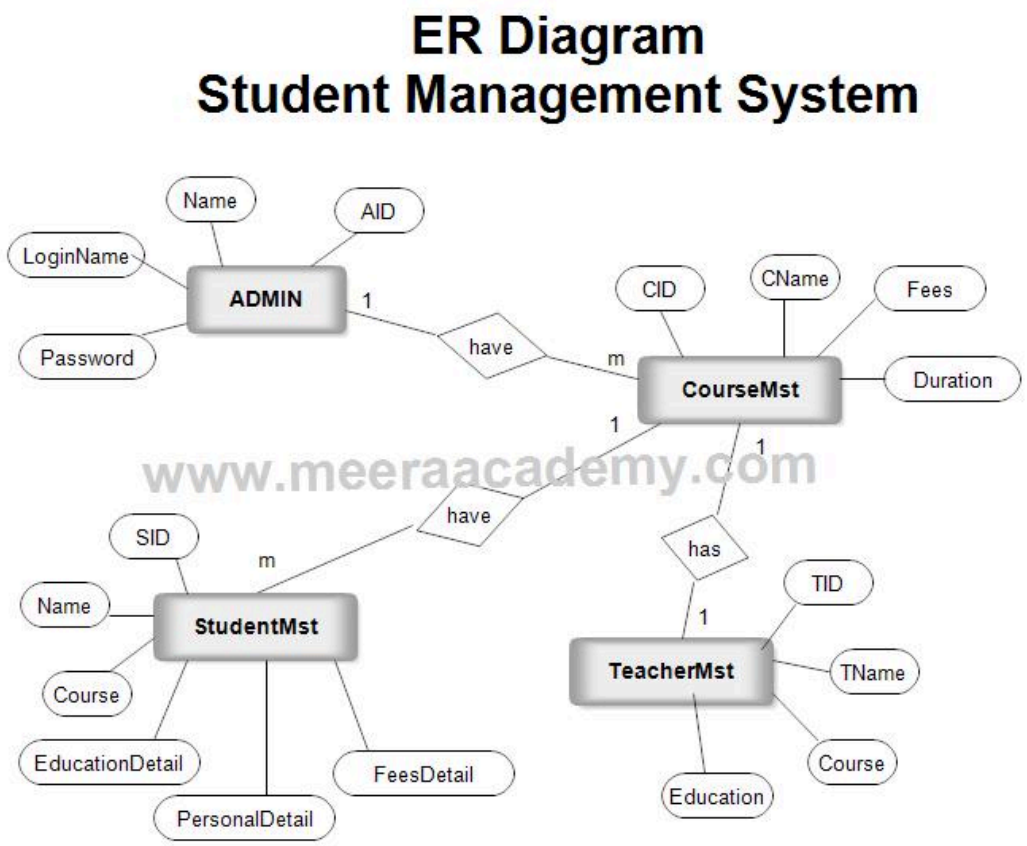
## Sequence Diagram




- **Student** submits complaint via web UI
- **Controller** validates input & saves record
- **System** returns confirmation to user

# Chapter 4: Detailed Design

## ER Diagram




Core entities: Student, Grievance, User (Admin/Staff)

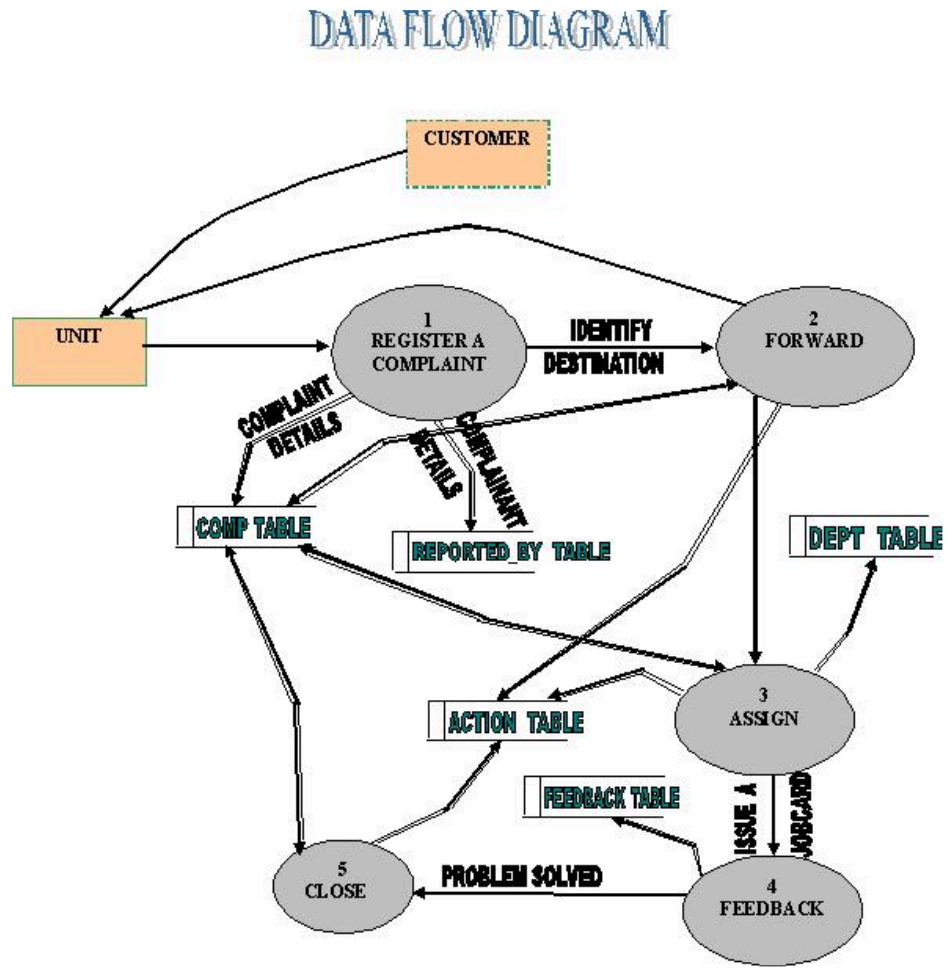
### Student

- studentID
- name
- email

### Grievance

- grievanceID
- title
- status

## Data Flow Diagrams




Data flow among external actors and processes

- **External entities**: Student, Admin, Staff
- **Processes**: Submit, Review, Update, Report
- **Data stores**: StudentDB, GrievanceDB

# Chapter 5: Implementation Plan

## Agile Sprint Breakdown

### 1 Requirements & UI

- Requirements gathering
- UI prototyping
- Login/register forms

### 2 Student Module

- Student dashboard
- Submission workflow
- Database models

### 3 Admin & Staff

- Review/assignment interface
- Status updates
- Role-based access

### 4 Testing

- Summary reports
- Validation
- System/integration tests

### 5 Deployment

- Production environment
- Final documentation
- User training

## CI/CD Pipeline

**Build**
Install dependencies
Static code checks

**Test**
Unit & integration tests
Code coverage

**Deploy**
Package application
Push to web host

**Rollback**
Monitor for errors
Restore if needed

## Infrastructure as Code (IaC) Plan

**VPC & Subnets**
Network configuration

**EC2/Elastic Beanstalk**
Django app hosting

**RDS Instance**
PostgreSQL database

**Security Groups**
Firewall rules

**S3 Bucket**
Static/media files

**Version Control**
Reproducible deployment

# Chapter 6: Quality Assurance Plan

## Testing Strategy

### Unit Testing

Test individual components (models, forms, utilities)

### Integration Testing

Test interactions between components

### System Testing

End-to-end testing in staging environment

### User Acceptance Testing

Representative users validate requirements

## Monitoring & Observability

**Logging** - Django framework for errors & events

**Metrics** - AWS CloudWatch for CPU/RAM usage

**Performance** - Request latency tracking

**Alerts** - Email/SMS notifications for critical issues

**Log Reviews** - Periodic analysis for early detection

## Chaos Engineering

### Purpose

Improve system resilience by testing failure scenarios

### Test Scenarios

- Database failure simulation
- Network partition testing
- Server restart verification

### Implementation

Scheduled drills during maintenance windows

# Chapter 7: Sustainability and Emerging Technologies

## Serverless & Modern Tech

### AWS Lambda
Notification service as serverless functions

### SPA Frontend
React/Vue calling Django REST APIs

### API Gateway
Microservices architecture for scalability

## Green IT Considerations

### Green Data Centers
Host in regions powered by renewable energy

### Resource Optimization
Scale down servers during off-peak hours

### Efficient Code
Optimize algorithms and use caching (Redis)

### IaC Portability
Deploy to eco-friendly cloud providers

## Future Enhancements

### Mobile App
Native app or PWA for easier access

### Chatbot Integration
AI-powered assistant for quick complaint intake

### AI/ML Analytics
Analyze trends and suggest resolutions

### Real-time Dashboards
Grafana integration for live metrics

# Chapter 8: Conclusion and Future Work

## Summary of Achievements

- **Core objectives** fully implemented
- **Django/SQLite** architecture justified
- **3-tier design** balances simplicity & scalability
- **QA & deployment** plans established
- **All requirements** addressed in design

## Future Roadmap

**Immediate:** Finalize deployment & UAT

**Short-term:** Migrate to PostgreSQL

**Mid-term:** Mobile app & enhanced notifications

**Long-term:** Real-time analytics dashboards

## Lessons Learned

### Technical

Django's ORM & admin interface accelerated prototyping

### Design

Clean data model prevented redesigns later

### Process

Agile approach enabled quick UI refinements

### Documentation

Clear requirements essential for success