**AFSPADE Technical Documentation**

**Table of Contents**

1. [Introduction](https://www.docstomarkdown.pro/convert-markdown-to-word/#introduction)
2. [System Architecture](https://www.docstomarkdown.pro/convert-markdown-to-word/#system-architecture)
3. [Technology Stack](https://www.docstomarkdown.pro/convert-markdown-to-word/#technology-stack)
4. [Database Models](https://www.docstomarkdown.pro/convert-markdown-to-word/#database-models)
5. [API Endpoints](https://www.docstomarkdown.pro/convert-markdown-to-word/#api-endpoints)
6. [User Authentication and Authorization](https://www.docstomarkdown.pro/convert-markdown-to-word/#user-authentication-and-authorization)
7. [Assignment Workflow](https://www.docstomarkdown.pro/convert-markdown-to-word/#assignment-workflow)
8. [Grading and Feedback System](https://www.docstomarkdown.pro/convert-markdown-to-word/#grading-and-feedback-system)
9. [Notification System](https://www.docstomarkdown.pro/convert-markdown-to-word/#notification-system)
10. [Deployment](https://www.docstomarkdown.pro/convert-markdown-to-word/#deployment)
11. [Testing and Quality Assurance](https://www.docstomarkdown.pro/convert-markdown-to-word/#testing-and-quality-assurance)
12. [Security Considerations](https://www.docstomarkdown.pro/convert-markdown-to-word/#security-considerations)
13. [Future Enhancements](https://www.docstomarkdown.pro/convert-markdown-to-word/#future-enhancements)
14. [Conclusion](https://www.docstomarkdown.pro/convert-markdown-to-word/#conclusion)

**1. Introduction**

AFSPADE (Automated Feedback System for Programming Assignments in a Digital Environment) is a web-based application designed to streamline the process of managing and grading programming assignments for universities, particularly Babcock University. The system allows lecturers to upload assignments, notify students, and automate the grading process, thereby reducing the workload on lecturers and providing timely feedback to students.

**2. System Architecture**

The AFSPADE system follows a standard web application architecture comprising:

1. **Client-Side (Frontend):** The frontend is responsible for rendering the user interface and handling user interactions. It communicates with the backend via APIs.
2. **Server-Side (Backend):** The backend handles business logic, database interactions, and API endpoints. It is built using Django, a high-level Python web framework.
3. **Database:** The database stores all data related to users, courses, assignments, submissions, and notifications. PostgreSQL is used as the primary database.
4. **Notification Service:** A service for sending email notifications to students regarding new assignments, submissions, and feedback.
5. **Automated Grading Engine:** This component is responsible for analyzing student submissions, identifying errors, and providing feedback.

**Diagram: System Architecture**

+---------------------+ +---------------------+

| Client-Side |<----->| Server-Side |

| (HTML/CSS/JS) | | (Django) |

+---------------------+ +---------------------+

| |

v v

+---------------------+ +---------------------+

| RESTful APIs | | Database |

| (Django) |<----->| (PostgreSQL) |

+---------------------+ +---------------------+

|

v

+---------------------+

| Notification Service|

| (Email API) |

+---------------------+

|

v

+---------------------+

| Automated Grading |

| Engine |

| (Python Rules/AI)|

+---------------------+

**3. Technology Stack**

1. **Frontend:** HTML5, CSS3, JavaScript, Bootstrap (optional)
2. **Backend:** Python, Django
3. **Database:** PostgreSQL
4. **Notification Service:** Django's Email Backend with SMTP / Brevo
5. **Automated Grading Engine:** Custom Python Scripts with rule-based and AI grading
6. **Version Control:** Git
7. **Deployment:** ?

**4. Database Models**

The system's data is structured using Django models, which are then translated into database tables.

**4.1 User Model**

Represents the users of the system (students and lecturers).

from django.contrib.auth.models import AbstractUser

class User(AbstractUser):

name = models.CharField(max\_length=255)

email = models.EmailField(unique=True)

phone\_number = models.CharField(max\_length=15)

matriculation\_number = models.CharField(max\_length=20, unique=True)

level = models.IntegerField()

semester = models.CharField(max\_length=15, choices=[('1st Semester', '1st Semester'), ('2nd Semester', '2nd Semester')])

is\_student = models.BooleanField(default=True)

def \_\_str\_\_(self):

return f"{self.name} ({'Student' if self.is\_student else 'Lecturer'})"

**4.2 Course Model**

Stores information about courses.

class Course(models.Model):

course\_name = models.CharField(max\_length=255)

course\_code = models.CharField(max\_length=20, unique=True)

lecturer = models.ForeignKey(User, on\_delete=models.CASCADE, limit\_choices\_to={'is\_student': False})

def \_\_str\_\_(self):

return self.course\_name

**4.3 Assignment Model**

Contains details about assignments uploaded by lecturers.

class Assignment(models.Model):

title = models.CharField(max\_length=255)

description = models.TextField()

upload\_date = models.DateTimeField(auto\_now\_add=True)

due\_date = models.DateTimeField()

course = models.ForeignKey(Course, on\_delete=models.CASCADE)

file = models.FileField(upload\_to='assignments/', blank=True, null=True)

question\_text = models.TextField(blank=True, null=True)

def \_\_str\_\_(self):

return self.title

**4.4 Submission Model**

Stores student submissions for assignments.

class Submission(models.Model):

student = models.ForeignKey(User, on\_delete=models.CASCADE, limit\_choices\_to={'is\_student': True})

assignment = models.ForeignKey(Assignment, on\_delete=models.CASCADE)

submission\_date = models.DateTimeField(auto\_now\_add=True)

file = models.FileField(upload\_to='submissions/', blank=True, null=True)

code\_text = models.TextField(blank=True, null=True)

grade = models.FloatField(blank=True, null=True)

feedback = models.TextField(blank=True, null=True)

def \_\_str\_\_(self):

return f"Submission by {self.student.name} for {self.assignment.title}"

**4.5 Notification Model**

Handles notifications sent to students.

class Notification(models.Model):

user = models.ForeignKey(User, on\_delete=models.CASCADE)

message = models.TextField()

date\_sent = models.DateTimeField(auto\_now\_add=True)

def \_\_str\_\_(self):

return f"Notification for {self.user.name} on {self.date\_sent}"

### **5. URL Routing and Views**

In this Django application, URL routing is used to map specific URL patterns to corresponding views. Each view processes user requests, interacts with the database if necessary, and returns the appropriate response, often in the form of rendered HTML templates.

**5.1 URL Patterns**

URL patterns define how different pages and functionalities of the application are accessed by users. These patterns are defined in Django’s urls.py file and determine which view is invoked when a user visits a specific URL.

**5.2 Views**

Views are responsible for handling the logic behind each page. They gather data from the models, apply business logic, and render the appropriate templates to be displayed to the user. Views can also handle form submissions, process user input, and return feedback or results.

**5.3 Templates**

Templates are HTML files that are rendered by views and sent as responses to the user’s browser. They can include dynamic content generated by Django's template language, allowing the pages to reflect data from the database or user-specific information.

**Forms and User Interaction**

Forms are used to capture user input and submit it to the server. The submitted data is processed by views, which may perform actions like authentication, data validation, or saving information to the database.

**6. User Authentication and Authorization**

**6.1 Authentication**

1. **Registration:** Users register by providing their name, email, phone number, matriculation number, level, and semester.
2. **Login:** Users log in using their matriculation number/email and password.

**6.2 Authorization**

1. **Role-Based Access Control (RBAC):** Users are assigned roles (student or lecturer) upon registration. Permissions are granted based on these roles.

**7. Assignment Workflow**

**7.1 Lecturer Workflow**

1. **Upload Assignment:** Lecturers can upload new assignments, specifying the course, deadline, and details.
2. **View Submissions:** Lecturers can view submissions made by students and download them if needed for offline review.
3. **View Grades:** Lecturers can see the grades assigned by the system to each submission.

**7.2 Student Workflow**

1. **Receive Notification:** Students receive an email notification when a new assignment is uploaded.
2. **Submit Assignment:** Students can submit their assignments by uploading files or pasting code directly into the platform.
3. **Receive Feedback and Grades:** After submission, students receive automated feedback and grades from the system.

**8. Grading and Feedback System**

**8.1 Automated Grading Engine**

The grading engine is responsible for analyzing the code submitted by students. The engine checks for:

1. **Syntax Errors:** Identifies any syntax issues in the code.
2. **Logical Errors:** Detects common logical mistakes (e.g., incorrect loop usage).

**Code Readability:** Checks for code indentation, variable naming, and comments.

1. **Documentation:** Ensures that the code is properly documented.

**8.2 Rule-Based Grading**

1. **Correctness:** Whether the code runs correctly and produces the expected output.
2. **Maintainability:** How well-structured and easy-to-read the code is.
3. **Efficiency:** The performance of the code in terms of time and space complexity.

**8.3 Feedback Mechanism**

The system provides detailed feedback on each submission, highlighting areas where the student can improve.

**9. Notification System**

**9.1 Email Notifications**

1. **Assignment Upload:** Sends an email to students when a new assignment is uploaded.
2. **Submission Confirmation:** Sends an email to students confirming their submission.
3. **Grade Release:** Sends an email to students when grades and feedback are available.

**9.2 Notification API**

The notification system is built using Django's email backend. It uses SMTP to send emails and integrates seamlessly with the rest of the system.

**10. Deployment**

**?**

**11. Testing and Quality Assurance**

**?**

**12. Security Considerations**

**12.1 Data Protection**

1. **Encryption:** Sensitive data, such as passwords, is stored using hashing algorithms (e.g., PBKDF2).
2. **Secure Connections:** All data transmission is secured using HTTPS.

**12.2 User Authentication**

1. **Session-Based Authentication:** Secure authentication using Django’s built-in session framework to manage user login sessions and protect views that require user authentication.

**12.3 Input Validation**

1. **Sanitization:** All user inputs are sanitized to prevent SQL injection and XSS attacks.

**14. Conclusion**

AFSPADE provides a comprehensive solution for managing and grading programming assignments in Nigerian universities. By automating many of the time-consuming tasks, it enables lecturers to focus more on teaching and provides students with timely feedback to enhance their learning experience.

This technical documentation provides a detailed overview of the AFSPADE system, including its architecture, models, APIs, and other components. It serves as a reference for developers and technical stakeholders working on the project.