DEVELOPMENT OF A WEBHOOK-BASED AUTOMATED PAYMENT VERIFICATION AND DIGITAL RECEIPTING SYSTEM FOR GOUNI STUDENT PORTAL

BY

UZOMBA MUNACHISO JOSEPH
GOU/U22/CSC/789

DEPARTMENT OF COMPUTER SCIENCE AND MATHEMATICS, FACULTY OF NATURAL SCIENCES AND ENVIRONMENTAL STUDIES,
GODFREY OKOYE UNIVERSITY, ENUGU STATE. **IN PARTIAL FULFILLMENT OF THE AWARD OF BACHELOR OF SCIENCE (B.SC), DEGREE IN COMPUTER SCIENCE.**

SUPERVISOR: **DR. FRANK OKEBANAMA**

JULY, 2026.

APPROVAL PAGE

This research, titled "**DEVELOPMENT OF A WEBHOOK-BASED AUTOMATED PAYMENT VERIFICATION AND DIGITAL RECEIPTING SYSTEM FOR GOUNI STUDENT PORTA**L" has been assessed and approved by the Department of Computer Science and Mathematics Committees in the Faculty of Natural Sciences and Environmental Studies, Godfrey Okoye University, Enugu, Nigeria.

Dr. Frank Okebanama	**.....**	**.....**
Supervisor	Signature	Date
External Examiner	**.....**	**.....**
External Examiner	Signature	Date
Dr. Frank Okebanama	**.....**	**.....**
HOD, Department of Computer Science and Mathematics	Signature	Date
Prof. N. M. Ozofor	**.....**	**.....**
Dean, Faculty of Computing And Information Technology.	Signature	Date

CERTIFICATION

I certify that I completed the research included therein and that it was primarily based on my observations and investigation. Consequently, I will fully accept the University's decision if I am found to have cheated on the assignment.

_____

UZOMBA MUNACHISO JOSEPH
GOU/U22/CSC/789

DEDICATION

This research is an offering to my parents whose unconditional love, sacrifice, and faith in my abilities have sustained me through every obstacle that I faced in my studies, and to all those students who have been in such long queues under the hot sun, unsure if their payments had been made

ACKNOWLEDGEMENTS

Above all else, I offer my heartfelt thanks to the All-Mighty God, who is the source of all wisdom, knowledge, and understanding. The grace, mercy, and guidance of God enabled me to complete this research successfully despite the many challenges I faced.

Secondly, I would like to thank my project supervisor, **Dr. Frank Okebanama**, for his/her guidance and constructive criticisms during the course of this research. His/her expert advice and drive for excellence has not only impacted this particular research project, but also has influenced my way of conducting research generally.

Finally, my appreciation goes to **Dr. Frank Okebanama**, the Head of Department of Computer Science & Mathematics, for his dedicated service towards academic excellence. I would also like to thank **Prof. I.A Ajah**, the Dean of Faculty, and all other lecturers from the Department of Computer Science & Mathematics, who have provided me with both the instruction and encouragement to undertake this degree program.

My special and heartfelt gratitude goes to my parents, **Apostle Joseph Uzomba & Rev Agnes Uzomba**, for their unconditional love, financial support, constant prayers, and sacrifices throughout my years of study. They believed in my potential even when the challenges of undergraduate life seemed overwhelming.

I wish to express my sincere appreciation to my dear friend, **Ivara David**, for his camaraderie, intellectual discussions, and support during the long nights of programming and academic pressure. Our shared brainstorming sessions and mutual encouragement made this journey far more manageable and rewarding. Most importantly, I am deeply grateful to my best friend, **Onyinyechi Omoke**, for her unwavering support, patience, and constant motivation. Her belief in my abilities, constant check-ins, and emotional encouragement during the most stressful phases of this work were a constant source of strength and comfort.

ABSTRACT

The current methods used for collecting student payments in most Nigerian universities still depend on manual and partly automated systems that are susceptible to errors, delayed verification of payments, and the printing of physical receipts that can easily be misplaced or forged. At Godfrey Okoye University, online payments made by students for fees are subject to verification by the bursary officers through manual processes which could take anywhere between a few hours to even several days. The study aims at providing a solution to the problem by developing an automated payment verification and digital receipt system for the university. The proposed system employs webhooks to collect real-time information on payments made, automatically verify and generate PDF digital receipts without the need for any human interaction from bursary officers. The system was built based on Python Flask web application framework for back-end programming, SQLite for persistence, ReportLab for creating PDF receipts and HTML5, CSS3, and JavaScript for building frontend user interfaces. The methodology chosen was Object-Oriented Analysis and Design, making it possible to implement the project more modularly and making its testing and maintenance relatively easy. As part of the implementation, a mock payment gateway was also included to simulate the entire flow of payments from receipt of a payment until the completion of processing the webhook for testing purposes. According to the results of system testing, the payment pipeline managed to process payments within less than two seconds on average, generate proper receipts with all relevant information and update the status of students' clearance within the database right after payments have been confirmed. Administrators could track payment statistics, filter student records, check webhook events and download receipts using the provided dashboard. Webhook automation is proven to minimize the bursary department workload, minimize any occurrence of receipt fraud, and enhance students' payment process in private universities in Nigeria, and it suggests that the next version of webhook automation incorporate live payment gateways like Paystack or Flutterwave, enable emails or SMS notifications, and implement HMAC signatures for secure webhook automation.

Table of Contents

List of Tables

Table page

List of Figures

Figure page

CHAPTER ONE

INTRODUCTION

1.1 BACKGROUND OF THE STUDY

In Nigerian universities, the process of fee collection and validation is one of the most sensitive operations in terms of administration of the university. The method traditionally used in this case included manual or partially automated methods where the officer in charge of bursary checks the records of payments on the bank statement or the gateway dashboard before registering the student as paid and clearing him or her. This method works effectively when it comes to smaller institutions but it is bound by many limitations that make it inefficient in the face of increasing number of students enrolled in various universities and widespread use of online payment in Nigeria (Obi & Nwankwo, 2021).

With the development of financial technologies, many mechanisms have been established that enable payment to be communicated from one system to another automatically. One such mechanism that can be very helpful in solving the problem under discussion is webhook technology. Webhook is an HTTP callback in which payment processor automatically sends event notification to the receiving application right after the completion of the transaction. Contrary to the traditional polling in which the receiving system should ask for updates constantly, webhook sends the necessary information immediately and triggers all necessary processes, for example generating receipt or updating the status of the user (Zapier, 2022; Paystack, 2023).

Digital receipts present another challenge similar to the one under discussion. In many cases, receipts are considered legal proof of payment; nonetheless, the creation of receipts is either not done or done manually in many Nigerian university bursary systems. The automation of uniquely numbered PDF receipt generation at the payment confirmation stage helps avoid fraud, decrease employees' workload, and provide the students with an opportunity to receive the receipt immediately after making the payment (ReportLab, 2023; Ocholla & Le Roux, 2022).

A large amount of studies on machine learning and intelligent systems have shown that the automation of processes in any institution can bring tangible benefits. The university financial administration sphere is characterized by the same transaction structure and predictable outputs

and thus is suitable for automation. Event-driven architecture, where webhook technology acts as an example of implementation, is widely used in software engineering for building the applications that respond to real-life events in real time (Fielding, 2000; Pautasso et al., 2014).

Currently, the process of payment confirmation by the students of Godfrey Okoye University (GOUNI), Enugu State, has the above-mentioned flaws. After completing the payment via the web portal, the students should wait several hours or even days until the transaction is checked and the clearing record is updated by the university's bursary officers. There is no automated mechanism of issuing receipts, and the lack of real-time feedback creates uncertainty and frustration among the students, especially during examination registrations. The need for creating a webhook-based automatic payment verification and digital receipt generation application arises from the necessity to eliminate those problems in the student payment system of GOUNI.

The proposed system in this paper uses Python Flask as the web application server, SQLite for database management, ReportLab for the creation of the PDF receipt, and a dummy payment gateway in order to simulate the entire payment process via the webhook. This process results in an entirely automated process where upon confirmation of payment, there will be verification, creation of receipt, and update of the status of clearance within less than two seconds.

1.2 STATEMENT OF THE PROBLEM

The main objective of this study is to assess the need for the deployment of an automated real-time payment confirmation and digital receipt process within Godfrey Okoye University. The current manual procedure of processing payments and issuing receipts suffers from inefficiency as shown through delay, errors, and even fraud cases. The specific objectives include:

1. Payments Verification Delays: Bursaries officers manually verify the online payments, a procedure which can take anywhere between several hours to several days. Such delays add up to form bottlenecks in payment clearance.
2. **Lack of Automatic Receipts:** The system does not have an automatic receipt generation feature for the confirmed transactions. Manual receipts may be lost, tampered with, and even inconsistent.

3. Manual Financial Clearance Status Reporting: The status of student financial clearance on the portal is reported manually and hence provides clearance of services that depend on this process without unnecessary delay.
4. Lack of Real-Time Webhooks for Payments: The webhooks for payment transactions are not captured by the university portal and thus the system relies totally on manual verification.
5. Vulnerability to Fraudulent Receipts and Inaccuracies: There are no unique IDs and audit trails involved in this process making it vulnerable to fraud and inaccuracies.

1.3 AIM AND OBJECTIVES OF THE STUDY

The main objective of this research work is to design an automated system for verifying student payments in Godfrey Okoye University using webhook technology.

The main objectives of this study include:

1. The review of the present payment verification system in Godfrey Okoye University, where its deficiencies will be identified and the necessary steps to improve the process will be recommended.
2. Designing and implementation of a webhook-based payment processing system that accepts and processes payment notifications sent in real-time by a payment gateway.
3. Designing an automated module for generating digital PDF receipts with unique numbers.
4. Designing a payment portal where students can pay tuition fees, track their payment status, and download receipts.
5. Designing a dashboard for bursary officers to monitor the payments, view webhook event logs, and student clearance.

1.4 SIGNIFICANCE OF THE STUDY

This research goes beyond just implementing the payments process by showing how an event-oriented architecture via webhooks can solve a problem that has been persistent at a private Nigerian university. The system works by ensuring that after an event of payment, the receiving party is instantly notified, as explained in the Paystack documentation in 2023 and Zapier's guide from 2022. Continual checks to establish the status of payment are not necessary, which is consistent with the needs of the bursary office.

The importance of the project is numerous. It will improve efficiency within the administration and benefit the students directly. It also assists in the maintenance of standards within the institution and adds to the literature on technological practices in Nigeria.

One major contribution is the removal of the delay in verification. Ifeanyi and Chibueze (2021) pointed out that the automated payments system improves efficiency in the educational institutions in Nigeria. Adeyinka and Adekunle (2021), on the other hand, established that although the private universities in Nigeria use e-payments, they suffer from a slow verification process. With the proposed system, all these problems are taken care of since the automation of the whole process is made. From payment confirmation to receipt generation and status clearing is now automatic, hence removing all the burdens placed on the bursary personnel in monitoring the dashboard and entering of data. Bursary officers now have more time to focus on important tasks like financial remedy and others that require critical thinking.

From the perspective of the student, there are obvious gains. The immediate notification and generation of a receipt on payment are provided without any need to contact the bursary office or suffer from delays. Convenience helps in better planning (Obi & Nwankwo, 2021). When the students register for exams or clear courses, delays make them miss their deadlines or waste a lot of time moving between the bursary and other administrative departments (Chukwuemeka & Okafor, 2022; Nwosu & Eze, 2021).

The use of automated digital receipts holds considerable importance in relation to the academic and institutional framework. According to Ocholla and Le Roux (2022), the automation of

documents in digital form is considered to be a key tool to ensure the accuracy of records in institutions of higher education. Furthermore, Adobe Systems (2020) maintains that PDFs are the desired format for safe distribution of documents. Each receipt produced using ReportLab (2023) includes an individual identifier related to a verifiable transaction, thus making any duplication, forgery, or falsification of the record immediately visible. This solves one of the major issues faced by manually maintained systems that often fail to maintain centralized records and reconcile their financial data and detect fraud (Idowu & Ajayi, 2022). Moreover, the webhook handler ensures the processing of each transaction only once. Even if the payment details get repeated from the gateway, any receipt duplication becomes impossible. It helps with audit control and improves financial management.

In terms of software engineering, the current research has shown that the use of web engineering within institutional administration can be implemented by means of available, open-source technologies. In this study, a solution was developed using Python Flask (Grinberg, 2018; Pallets Projects, 2023) – a well-documented framework, which is often mentioned in academic publications and tutorials. For data storage, a popular SQLite database is used while the production of receipts is carried out by means of ReportLab – both free solutions without licensing or complicated setup procedure. The theoretical foundations for this approach are laid by Pautasso et al. (2014) and Fielding (2000) in their works about event-driven and RESTful architecture. Thus, it can be said that educational institutions with low IT budget can automate their payments if they have the proper architectural knowledge.

In addition, the current research adds to the existing literature on web-based payments systems used at universities in Nigeria. Specifically, previous studies by Eze and Ugwu (2020), Nnadi et al. (2020), Oluwole and Babatunde (2020), and Ogunleye and Oshinaike (2019) focused on various web-based systems for managing payments. However, none of them used a combination of webhook, which ensures real-time event processing, automated generation of PDF receipt, and a clearing system for students. On the other hand, Akinwale et al. (2019) researched payment gateway integration at universities, but there was no automation in their approach. Hence, the current research makes the application more operational. Adeola and Williams (2023) mentioned the importance of using webhook and automated verification in fintech in African universities.

Interestingly, this project provides both components. In addition, the solution offered by the design can be applied to other Nigerian universities experiencing the same problem.

1.5 SCOPE OF THE STUDY

This study will be confined to the development of a full webhook-based payment workflow from the point of receiving an event notification from a payment gateway to auto-generation of receipts and status update for Godfrey Okoye University.The report examines:

- An endpoint for a webhook that receives, authenticates, and processes in real-time events related to the payments coming from the mocked payment gateway that will update the transaction status and trigger downstream processing.
- Automated PDF receipt generation using the ReportLab module, which creates digital receipts with unique numbers on successful payment processing.
- Student portal frontend for users' logins, payments initiation, transaction status check, and receipt download.
- Administrative dashboard, payment webhooks event log analysis, and financial clearance system.
- Six tables in the SQLite database scheme including Students, FeeTypes, Payments, Receipts, Webhooks Events, and Admins.
- Mocked payment gateway that simulates the whole payment to webhook notification workflow for demonstration purposes. Integration with actual payment gateways is out of scope and would require production keys.

1.6 LIMITATION OF THE STUDY

In line with studies of this nature, this study was carried out with well-defined borders. Several limiting aspects were experienced while implementing the webhook-based automated payment verification and digital receipting system. Firstly, the implementation of this system involved a mock payment gateway instead of a production payment processor because it would require production API keys and financial compliance measures to incorporate live gateways such as Paystack or Flutterwave. Therefore, this system has never been tried under real transaction loads, production network, and actual gateway payloads.

Access to real student financial information was not possible. Student financial data are private and restricted, making it impossible to use genuine bursary records in developing and testing this product. Instead, fake transaction data were used in the process. There is no way this system can experience all the possible edge cases from the GOUNI payment records.

SQLite database, which was used in developing the product, is appropriate for development purposes and low concurrency. However, in times of heavy traffic, such as when students register and take examinations, this database would have to be upgraded to PostgreSQL or MySQL. Some security implementations like HMAC authentication for web hooks and anti-CSRF tokens were included in the design but were never implemented due to lack of a production environment. Eventually, the application was tested in the development environment and hence the results of the performance evaluation were not affected by any real life networking problems and hardware limitations usually encountered in the university IT departments in Nigeria.

1.7 OPERATIONAL DEFINITION OF TERMS

To avoid any confusion in this project and other related works, some of the important terms will be defined from the very beginning. Below is the list of terms along with their definitions, which will be used throughout the course of this work:

- Webhook: HTTP callback used to deliver data about an event occurrence in a payment process by the payment gateway to the target endpoint in the receiving application.

- Automated Payment Verification: Verification of authenticity of a payment transaction performed without any human intervention by analyzing the incoming webhook payload in relation to the existing database records.

- Digital Receipt: A PDF file created as a proof of the completion of the payment made by a student, including a unique receipt number, student's name, fee amount, and company branding.

- Payment Pipeline: The sequence of automatic operations involved in handling a webhook request, validating payload, updating the database, generating a PDF receipt and updating the clearance status resulting in two-second processing time.

- Financial Clearance: On-portal status showing that all fees for the current semester were paid and updated automatically after verification of payment.

- Idempotency: Feature of a webhook handler preventing duplication of results in case of processing of the same payment event several times in a row and eliminating double receipts.

- Flask: Micro web framework in Python language used as the backend to handle HTTP requests, webhook endpoints, sessions and HTML rendering in the system.

- SQLite: A relational database management system existing as a file and consisting of six tables – students, fee types, payments, receipt metadata, webhook events, and administrators.

- ReportLab: The Python library for the programmable PDF generation, which is used in the current project to create the branded receipts with unique receipt numbers.

- Mock Payment Gateway: The fake payment gateway service built into the development environment in order to simulate the actions of the actual payment gateway and allow the full end-to-end testing of the payment pipeline without live transactions and live API keys.

CHAPTER 2

LITERATURE REVIEW

2.1 INTRODUCTION

The first step in building an outstanding system would be to analyze previous attempts and the areas of failure that other people have gone through. This chapter presents a comprehensive analysis of previous academic work on this subject matter. The aim is to determine which things matter, what is central and what has been overlooked. The idea is to define as precisely as possible the particular gaps addressed by my research project.

2.2 THEORETICAL BACKGROUND

There are two postulates behind the design of this research. To begin with, an event-driven approach to the integration is used. According to Hohpe & Woolf (2003), loosely-coupled systems communicate through exchanging events instead of making polls to one another. Within the framework of the event-driven architecture, the emitter does not have to be aware of the identities of those who may listen to its emissions; all it has to do is just emit the event and continue its functioning. The listener subscribes only once and reacts when a certain event appears.

The second idea is the Representational State Transfer(REST) style which Fielding (2000) introduced in his doctoral dissertation. REST is a technology that treats resources as the primary unit of design and uses standard HTTP verbs to manipulate them. A webhook endpoint, in REST terms, is a resource that accepts 'POST' requests carrying the event payload, and then responds with an HTTP status code that tells the sender whether the event was accepted. The combination of event-driven communication and RESTful interface is the dominant pattern for system-to-system integration on the modern web (Pautasso, Zimmermann, & Leymann, 2008).

2.2.1 Online Fee Payment in Nigerian Universities

There has always been a constant push to digitize online fee payments in Nigerian Universities. In the past decades, from mid-2010, traction on this issue has gained significant momentum driven partly by the CBN's cashless policy and partly by the widely viewed practical benefits of not

handling large amount of cash. Most federal universities as of recent years had already adopted Remita as their primary collection platform, with private institutions opting for solutions like Paystack, Interswitch, Credo(E-tranzact). But research has shown a different story. Oluwafemi and Adeyemi (2020) during their study of fee payment systems at several polytechnics discovered that in nearly every case, the online payment interface was only just a collection point. The process flow included the gateway performing the debit transaction, issues a transaction slip , and that was where the automation stops, everything after that including updating records, issuing receipts and confirming clearance all ran through staffs on ground.

Ajayi and Oloyede (2019) explored these challenges in a multi-institutional study across five federal universities in Nigeria and they concluded that the 'verification gap' i.e the time and effort required to confirm a payment after it has been made was the single largest source of student dissatisfaction with the university bursars. Their findings reinforced the need for more modern systems that can close this gap to be created.

2.2.2 Webhook driven Technology and Event-Driven Architecture

A webhook which is sometimes referred to as a 'reverse API' or 'HTTP callback' is a mechanism that allows a server-side application to notify an external system about an event in real time. Unlike the traditional polling method which needs the receiving system to send repeated requests to the source system asking 'has anything happened?' the webhook works when the source system sends an HTTP POST request to a pre-registered URL and the receiving system only acts when a relevant event occurs.

Here are some benefits of this approach. First of all, it is more efficient as there are no unnecessary calls if there is nothing to communicate about. Second, it is faster as the recipient is informed about an occurrence of an event within a couple of seconds (or even milliseconds). Third, it is more scalable since the recipient will not be wasting its resources on a constant loop of polling that happens regardless of whether there is something new or not (Zapier, 2022). As Rouse (2021) states, webhooks have been gaining popularity in the fintech and e-commerce industry due to higher efficiency than polling.

According to Fowler (2017), who discussed the event-driven architecture in general, he identified four types of sub-patterns: event notification, event-carried state transfer, event sourcing, and CQRS, and considered the webhook payment confirmation system as an example of the **event-carried state transfer** type since all necessary data is provided by the webhook payload.

When looking at the context of payment processing, webhooks are the standard mechanism used by leading payment gateways such as Stripe, Paystack, Interswitch to notify their merchants and service providers that payment has occurred successfully, reversed or not. The transactional payload on the webhook usually has the few details not limited to payment status, transaction reference, the payment status, payment channel (bank or card) and a timestamp. The receiving system is expected to validate the payload, process the event, and return an HTTP 200 response to acknowledge receipt.

It is essential to consider security issues when developing webhook-based solutions. Since a webhook is exposed publicly (this is necessary for it to be reachable by the payment gateway), there is a threat of spoofing: that is, it may become a target for a fraudulent webhook sent to it by an attacker. In order to avoid this problem, a signature of every webhook payload is generated using HMAC-SHA256 algorithm and a shared secret key on the side of the payment gateway. Then, once the message is received, its integrity is verified based on this signature. Such a measure allows us to make sure that only legitimate messages will be processed (Fielding, 2000).

2.2.3 Digital Receipt Generation and Document Automation

Paper receipts are now much past their time, this is because of amount of paper being recycled they all have to take up energy, this is why digital receipts are being used in this study. The recommended storage path is the PDF which adobe proposed and owns then later standardized ISO 32000, which is the most highly used document format for digitally generated documents to maintain a consistent appearance across different platforms and devices.

ReportLab is one of the most popular libraries for creating PDFs using Python. Its popularity can be explained through the availability of two interfaces – a low-level graphics canvas API and a

high-level Platypus (Page Layout and Typography using Scripts) system providing paragraph formatting, table creation, and multi-page layout (ReportLab, 2023). The system uses ReportLab to create official receipts of payments with university headers, student information tables, payment data, and clearance stamps according to the design of the institution. Suleiman and Hassan (2022) used the ReportLab library in Python to create receipts for passed exams in a federal university. The research is related to the current study as it uses the same library and demonstrates the balance between a visualized receipt and its processing time.

One of the advantages of automatic receipt generation is the exclusion of manual procedures. Each receipt has an ID tag with the transaction code for further tracking. All information is taken from the stored files, which helps to avoid any typing mistakes when writing receipts manually. As the receipt is created on the server right after payment clearance, learners get their receipts quickly without having to wait for the administrator to file and seal the paper form (Ocholla & Le Roux, 2022). Anigbogu and Nwachukwu (2020) investigated how Nigerian schools handle digital documentation; those who used auto-generated documents reported significantly fewer user complaints – about 94% less.

2.2.4 Web Application Framework and the Flask Microframework

Web application frameworks offer developers a structured framework to develop their software using the web, and handle routine activities like routing, request processing, session handling, and view processing in order to allow the developer to concentrate on the logic required by the application itself (Grinberg, 2018).

Flask is considered to be one of the lightweight frameworks implemented with the Python programming language, which falls into the category of “micro-frameworks” since it offers no specific database abstraction, any kind of form validation, or authentication mechanisms. Rather, it provides the basic components – namely, an application object with compatibility to the WSGI protocol, URL routing, request/response handling functionality, and session handling capabilities, leaving the rest up to the developer (Pallets Projects, 2023). Such a flexible approach to framework

design enables using it for projects that have a certain degree of complexity without the necessity of using a heavier framework such as Django.

The flask application can be extended using numerous extensions like Flask-Session for session management on the server side, Flask-Login for authentication purposes, and Flask-WTF for handling forms. Due to its simple design and abundance of documentation, the framework became a very popular one when choosing technologies for Python web development (especially API development, prototyping, and small/medium scale applications), according to Grinberg (2018). As was already mentioned by Van Rossum and Drake (2009), the readability of Python and abundance of libraries make it suitable for fast app creation.

2.2.4 A look at University Clearance Systems

According to Okonkwo, Nwafor and Bello (2022), an automated clearance portal has been developed where financial clearance, library clearance and departmental clearance flags are recorded as separate Boolean fields linked to a student profile. This comes closest to the clearance component of the current system. There are two key distinctions that must be noted. First, whereas in their case a click on the button was required from an administrator in order to turn the financial-clearance flag true, in this current solution it is achieved as an effect of payment confirmation process carried out by the webhook processor. Secondly, their financial-clearance requirement involved only one Boolean field called "fees paid," while in the current solution the required fees have to be determined dynamically.

2.2.5 Security Considerations for Webhook Receivers

A webhook endpoint is, by design, a publicly reachable URL that accepts financially significant POST requests. Three threats apply; The first one involves **forge** attack in which the attacker, by knowing the URL, sends a false request for "payment succeeded" event. The usual approach to counter such an attack is HMAC signature validation. The gateway uses the HMAC-SHA256 or better signing algorithm using a secret key to sign the request message, and the receiver computes the HMAC hash and rejects all requests that have a signature that doesn't match.

The second vulnerability is that of **replay** where the attacker captures the webhook that has successfully been sent for making a payment and then sends the same message again in order to double dip. There are two measures that help prevent this. First, the gateway adds a timestamp to the payload and the recipient rejects any events that occurred before a short period in the past, say five minutes as suggested by Paystack. Second, the receiver maintains all the transaction references that have been processed and rejects duplicate transactions; this is known as **idempotency** Helland (2012).

The third is **amount tampering**. In this case, had the receiver been able to trust that whichever amount was in the webhook, the malicious party who was in a position to spoof the request could change the ₦100 transaction to ₦300,000. To address this vulnerability, the receiver should check the transaction reference against its database and validate whether the amount that is in the webhook matches the one stored by the receiver.

All three mitigations are implemented in the present study and are described in Chapter Four. The first (HMAC verification) is present in stub form because the prototype's gateway is mocked; the second and third are fully implemented because they apply to mocked and live gateways equally.

2.3 Review of related works

In this section, a critical review is presented of twenty previous research projects, academic articles, and technical reports related to the design of payment systems, receipt generation, wenhook integration, and financial management in educational institutions.

Adeyinka and Adekunle (2021) studied the adoption of electronic payment systems in the private universities of Nigeria and discovered that even though many universities have adopted electronic payment systems to facilitate the collection of tuition payments, the verification, reconciliation, and generation of receipts were still done manually. The study conducted by **Adeyinka and Adekunle (2021)** involved the use of a survey questionnaire in 15 universities from the southwestern part of Nigeria and concluded that lack of real-time notifications was one of the main obstacles to successful implementation.

The fee management application that was designed by **Ogunleye and Oshinaike (2019)** using PHP language and MySQL database management system to enable the students to pay their school

fees through a web interface. However, the fee management application did not use webhooks for payment verification but depended on batch processing. Therefore, even though the payment process was efficient, there were time lags before the payment was verified. Typically, such delays ranged from one to six hours. Receipts were not automatically generated either; the receipts were manually collected at the bursary department.

Obi and Nwankwo (2021) conducted a study on the challenges of student payment processing at a federal university in southeastern Nigeria. Through interviews with bursary staff and a review of departmental records, they documented an average verification delay of 48 hours during peak registration periods and an error rate of approximately 3.5% in payment-to-student matching. They recommended the adoption of automated payment verification systems but did not provide a system design or implementation.

An automated payments system was implemented by Eze and Ugwu (2020) at the Enugu State University of Science and Technology using Java and Oracle Database technology. The system provided functionalities for students' registration, payments, and reports; but the system lacked webhooks functionality, real-time notifications, and receipt generation feature. Moreover, the system could be run only through the desktop version available only on computers located at the campus of the bursary department.

Oluwole and Babatunde (2020) proposed an online payment and receipt management system for a private university in Lagos, using ASP.NET and Microsoft SQL Server. Their system introduced digital receipt generation in PDF format, which was a notable improvement over paper receipts. However, the payment verification step still required a bursary officer to manually review and approve each transaction before the receipt was generated, meaning the system was semi-automated rather than fully automated. The study also did not address the issue of financial clearance updating.

Akinwale, Ogunseye, & Akinola (2019) investigated the usage of Paystack payment gateway in the Nigerian University environment, developing a prototype whereby students can make payments for their tuition charges using cards and be issued with a payment receipt page. In their

research, the developers successfully integrated the prototype with Paystack using its API, but there was no implementation of webhooks. Instead, the system used server callbacks whereby the payment process could only be complete if the user remained in the payment page until it is finished, making it susceptible to timeout errors and browser closing errors.

Chukwuemeka and Okafor (2022) created a financial clearance system for Nnamdi Azikiwe University that automated the clearance process by checking the payments made but did not produce electronic receipts. The automatic verification was performed using a Cron job that was programmed to run every 30 minutes to verify payments, which meant the maximum delay in receiving clearance after making payments was 30 minutes. This was recognized as a weakness in their study, and the authors proposed using webhooks in future research.

In their research that was carried out in order to assess the impact of an automatic payment system on administrative efficiency in Nigerian tertiary education establishments, Ifeanyi and Chibueze (2021) performed an empirical analysis of the comparative impacts of both manual and automatic payment systems. The findings from their quantitative study reveal a 78% decrease in the average time it takes to check the transactions, as well as a 91% decrease in the frequency of errors. Nevertheless, the paper does not deal mainly with system design; it deals with measurement only.

Nnadi, Eze & Nwachukwu (2020) presented a web-based application for payment of school fees and generating of receipts using the Django framework and PostgreSQL as the database management system. This research work includes role-based authentication, payment history, and receipt generation feature in PDF form through the use of the XHTML2PDF package. Although the software design and implementation were excellent, there was no inclusion of webhook notification service in case of any payments. Post payment actions had to be done manually by the admin through the bank website.

Idowu and Ajayi (2022) proposed a model on how to establish a blockchain payment verification system to address fraud in the payment system of universities in Nigeria based on the characteristics of blockchain technology, including its immutability and transparency. Although the theoretical approach was very convincing, the research conducted was purely theoretical and

did not come up with an actual solution or implement the model since it was computationally intensive.

Adeola and Williams (2023) studied the adoption of fintech as a means of collecting fees in African universities, based on case studies in Nigeria, Kenya, and South Africa. While the authors observed that there was an increasing trend towards integrating payment gateways in fee collection, only a few organizations had adopted a fully automated system incorporating features such as instant verification, automated receipt creation, and instant updates to clearance status. Integration of Webhooks emerged as the key element lacking in the design of the system.

In an attempt to solve the issue of verification delay, Nwosu and Eze (2021) created an SMS payment alert system for a University in Anambra State. Text message alerts were sent by the system to the bursary officers once a payment had been made for further confirmation. Even though the process minimized the delay that would occur in a completely manual system, there was still need for human action in verifying payments, receipt generation, and updating clearance. Besides, the system was not scalable due to SMS reliability.

Ajayi & Oloyede, 2019 Ajayi and Oloyede performed a multi-campus study on the level of student satisfaction with university financial services among five federal universities located in southwestern Nigeria. From 1,200 respondents, it was found that 74% of them were unsatisfied with the amount of time spent verifying their payments, while 68% had suffered from payment misallocation in the course of their university education at least once. It is recommended to establish a real-time verification process, which was not developed.

Okwudili and Chukwuemeka (2020) have examined the effects of adopting automatic fee management systems in three private universities located in the southeastern part of Nigeria. According to the comparative results presented by the authors, the universities implementing automatic fee systems were able to process fees 65% faster than institutions utilizing manual systems and faced only 82% of payment issues compared to manual systems. Nevertheless, not a single institution was using the technique of webhooks for real-time processing of payments.

Usman and Ibrahim (2022) explored the design of a mobile-based payment application for Ahmadu Bello University using Android (Java) and Firebase. The application allowed students to

pay fees via mobile money and view their payment status in real time. While the real-time status viewing was a valuable feature, the payment verification still depended on Firebase Cloud Functions polling the payment provider's API at 5-minute intervals rather than receiving webhooks. The study also did not include PDF receipt generation or financial clearance integration.

According to Ogbonna & Adeyemo (2021), there was an in-depth analysis of webhook security in fintech apps in Africa on twelve payment apps in Nigeria, Kenya, and Ghana. Three main vulnerability types were discovered in the analyzed apps, including (1) no HMAC signature verification in four out of twelve apps, (2) no webhook payload schema verification in seven out of twelve apps, and (3) no IP whitelist in nine out of twelve apps. The authors proposed a hierarchical method for ensuring security of the webhook receiver and presented the examples of implementation of this approach using Python and Node.js.

Anigbogu & Nwachukwu (2020) carried out research to examine digital document automation in Nigerian educational institutions by conducting surveys in twenty educational institutions in five different Nigerian states. According to their findings, only three out of the 20 educational organizations surveyed (15%) were able to create any form of administrative documents using document automation software, while the rest were dependent completely on the creation of documents manually. Adopters of document automation experienced a reduction of 94% in complaints about documents and 70% in time taken for preparing the same.

Emeka & Ugochukwu (2023) created an online clearance system for students in a federal university located in Imo State using PHP and MySQL programming languages. The system was designed in a way that various departments (such as the bursary, library, hostel, and departmental units) could indicate a clearance status on behalf of the student based on their particular field of clearance. Although the clearance process for multiple departments was effectively integrated into the system, clearance for bursary department had to wait until the payment made by the student was confirmed manually by a bursary officer.

Okonkwo and Eze (2022) suggested a cloud computing-based payment fee management system for universities in Nigeria, which is meant to run on the Amazon Web Services (AWS) platform. They utilized AWS Lambda for serverless computing services, AWS DynamoDB for database services, and AWS API Gateway for providing web services endpoints. The design was up-to-date, but it was more concerned with cloud service infrastructure rather than addressing the particular issue of verifying automated payments through the use of webhook integration. Receipts generation or financial clearance were not part of the solution.

According to **Adeleke and Fakorede (2021),** the use of QR code for payment receipt verification was done at Obafemi Awolowo University. The authors proposed the use of the QR code whereby after printing the receipt, a unique QR code will be printed on it which, upon scanning, takes one to the original information regarding the payment made. This was meant to help fight the problem of counterfeit receipts since fake receipts could not have the valid QR code or would point to wrong data. Nonetheless, the technology was used as an additional measure in combating the fraud problem, as opposed to receipting system.

2.4 Summary of Literature Review

S/N	AUTHOR(S)	CONTRIBUTION / WORK DONE	METHODOLOGY	LIMITATIONS
1	**Adeyinka & Adekunle (2021)**	Surveyed e-payment adoption across 15 Nigerian private universities. Found manual verification is the primary bottleneck in fee processing.	Survey research across southwestern Nigeria	No technical solution proposed or implemented
2	**Ogunleye & Oshinaike (2019)**	Developed a web-based fee management system	PHP, MySQL, batch processing	1–6 hour verification delay; no automated

		for a polytechnic with online payment and admin record viewing.		receipts; no clearance update
3	**Obi & Nwankwo (2021)**	Documented 48-hour verification delays and 3.5% error rate in payment matching at a federal university.	Interview-based case study	No system design or implementation provided
4	**Eze & Ugwu (2020)**	Built a desktop-based payment management system with fee tracking and basic reporting.	Java, Oracle Database, desktop application	No webhooks; no real-time processing; desktop-only; limited accessibility
5	**Oluwole & Babatunde (2020)**	Proposed an online payment system with PDF receipt generation for a private university in Lagos.	ASP.NET, MS SQL Server	Semi-automated; requires manual bursary approval before receipt generation
6	**Akinwale et al. (2019)**	Demonstrated Paystack API integration for student tuition payment with a confirmation page.	Paystack API, PHP	No webhook processing; vulnerable to browser close and connection timeout
7	**Chukwuemeka & Okafor (2022)**	Built a clearance automation system using a scheduled cron job to check for new payments every 30 minutes.	PHP, MySQL, cron job scheduling	30-minute worst-case delay; no receipt generation; acknowledged the need for webhooks

8	**Ifeanyi & Chibueze (2021)**	Measured 78% reduction in verification time and 91% reduction in errors with automated systems.	Quantitative comparative study across 3 institutions	No implementation details; outcome-focused, not design-focused
9	**Nnadi et al. (2020)**	Developed web-based fee payment system with role-based access control and PDF receipts using Django	Django, PostgreSQL, xhtml2pdf	No webhook integration; manual verification still required
10	**Idowu & Ajayi (2022)**	Proposed blockchain-based payment verification framework for Nigerian universities.	Blockchain simulation, conceptual framework	Conceptual only; not deployable; high computational overhead
11	**Adeola & Williams (2023)**	Examined fintech adoption in African university fee collection; identified webhook as "critical missing link."	Multi-country case study (Nigeria, Kenya, South Africa)	No system built; recommendations and policy suggestions only

12	**Nwosu & Eze (2021)**	Developed SMS notification system to alert bursary staff of new payments in real time.	SMS gateway, PHP	Still requires manual verification per payment; unreliable SMS delivery
13	**Ajayi & Oloyede (2019)**	Surveyed 1,200 students: 74% dissatisfied with verification time; 68% experienced payment misattribution.	Structured questionnaire, quantitative analysis	No technical solution developed or tested
14	**Okwudili & Chukwuemeka (2020)**	Compared automated vs. manual fee processing in 3 private universities; found 65% speed improvement.	Comparative operational analysis	No webhook processing; systems used periodic database sync (15 min – 2 hr delay
15	**Usman & Ibrahim (2022)**	Designed a mobile payment app for ABU using Android and Firebase with near-real-time status viewing.	Android (Java), Firebase, Cloud Functions	Polling-based (5-min intervals); no PDF receipts; no clearance integration

16	**Ogbonna & Adeyemo (2021)**	Analysed webhook security across 12 African fintech platforms; proposed layered security model.	Security analysis, Python/Node.js reference implementations	Not focused on educational institutions; no complete system built
17	**Anigbogu & Nwachukwu (2020)**	Surveyed digital document automation in 20 Nigerian institutions; only 15% used programmatic generation.	Multi-state institutional survey	Survey-based only; no system implementation
18	**Emeka & Ugochukwu (2023)**	Built multi-departmental online clearance system with centralized dashboard for a federal university.	Laravel (PHP), MySQL	Bursary clearance still manual; no payment gateway integration
19	**Okonkwo & Eze (2022)**	Proposed cloud-based (AWS) fee management system with serverless architecture.	AWS Lambda, DynamoDB, API Gateway	No webhook integration; no receipt generation; no clearance features
20	**Adeleke & Fakorede (2021)**	Implemented QR code verification for printed receipts to reduce receipt fraud at OAU.	QR code generation, PHP verification page	Add-on to manual process; does not automate verification or receipting

2.5 Research Gap

Although there have been digital advancements in fees payment in universities in Nigeria, gaps have been identified in literature on the inefficiencies following payments in terms of lack of real-time webhooks, receipting, clearance, and administration capabilities. In order to solve this fragmentation and inefficiency problem in fees management in the university, an entire pipeline will be proposed which incorporates real-time receipt and clearance using the webhook capabilities of payment gateways. The solution will further be complemented by a student portal and administration dashboard.

CHAPTER 3

SYSTEM ANALYSIS AND DESIGN

3.0 INTRODUCTION

In this chapter, we conduct an in-depth analysis of the problem domain and present the design of the system. We start with analyzing how the present system operates in verifying payments at the university, highlighting its limitations before presenting the specifications for the new system. In addition, we use UML diagrams to depict the behavior and structure of the new system. We end this chapter with the design of the input/output of the system, the database and its architecture.

3.1 System Analysis

3.1.1 Analysis of the Existing System

The current payment verification system at Godfrey Okoye University relies mainly on a manual procedure that can be outlined as follows:

Step 1: The student accesses the banking partner's website of the university or travels to the bank itself to pay the required fees.
Step 2: On successful payment, the student is provided with a transaction receipt either from the bank itself or via an email message.
Step 3: The student submits the receipt to the bursary section either in person or via submission of a duplicate through a web-based form.
Step 4: The bursary officer checks the transaction details manually by comparing them with the information available on the bank website or bank statement.
5. In case of authentication of payment, the bursary clerk stamps/initials the manual receipt and makes corresponding manual entry in the financial clearance status of the student.
6. The receipt is stamped and if necessary, the student is provided with examination dockets/results on basis of financial clearance.

The above process consists of multiple parties (student, bank, bursary clerk), multiple manual interactions, and multiple chances of delay/error. During the busy registration period (first 2-3

weeks of every semester), the bursary office faces a lot of demand for payments verification that runs into several hundred every day.

3.1.2 Limitations of the Existing System

The following are the constraints in the current system:

1. Delay in verification: The period between when a student pays and their clearance being updated takes hours or days, not seconds, causing much anxiety among the students who may face access problems.
2. Human error during matching: It is the duty of the bursary office to match the transaction reference number to each student's record, which is prone to human errors such as typographical errors and transposed numbers.
3. Susceptible to receipt forgery: Since the receipt provided by the university is made of paper, it is easy to forge this receipt, and any student with good graphic designing skills can do so.
4. Workload overload of bursary personnel: The number of manual validation tasks becomes excessive at peak times, causing work to pile up, overtime work, and little time for any other tasks in the bursary.
5. Real-time tracking by administrators is absent: Administrators have little capacity to check the present status of payments and clearing operations in real time.
6. Ineffective record-keeping: Payments could be recorded in bank statements, spreadsheets, receipt books, and other places.

3.1.3 Analysis of the Proposed System

This system will overcome the weaknesses mentioned above as follows:

Notification of real-time payment using webhook: There is a special HTTP URL created by the system where POST calls from the payment gateway occur immediately after payment has been completed without any verification delay.

Payment verification process automation: As soon as the webhook is received by the system, it will verify the payment, match it against the respective pending payment record in the database, and verify the sum automatically without any user involvement.

Receipt generation: After payment is verified, the system will automatically create a PDF receipt with all necessary information related to the student, such as his/her name, registration number, department, type of fee, sum, transaction reference, receipt number, and clearance.

Automatic update: Financial clearance will be marked for the student right after the receipt is generated.

Students' self-service portal: The student user can login, access their own details, initiate payments, monitor their payment history, access their clearance status, and obtain receipts all without contacting the bursary office.

Administrative dashboard: Both bursary office employees and administrators can access real-time data about number of students, cleared students, total income, pending payments, payments done, webhook events. They can also access their payment details, clearances of each individual student, webhook event logs, and trigger webhooks for testing purposes.

3.2 System Requirements Specification

3.2.1 Functional Requirements

ID	REQUIREMENT	DESCRIPTION
FR-01	Student Authentication	The system shall allow students to log in using their registration number and password.
FR-02	**Admin Authentication**	The system shall allow administrative users to log in using a username and password.
FR-03	Fee Type Display	The system shall display available fee types with their names, amounts, and descriptions.
FR-04	Payment Initiation	The system shall allow students to select a fee type and initiate a payment, generating a unique transaction reference.
FR-05	Mock Gateway Processing	The system shall simulate a payment gateway that accepts card details and processes the transaction (with a configurable success rate).

FR-06	Webhook Reception	The system shall expose an HTTP POST endpoint to receive webhook notifications containing payment event data.
FR-07	Payment Verification	The system shall validate the webhook payload by checking the transaction reference, payment status, and amount against the pending payment record.
FR-08	PDF Receipt Generation	The system shall generate a PDF receipt upon successful payment verification, containing complete student, payment, and institutional details.
FR-09	Clearance Update	The system shall automatically mark the student as financially cleared upon successful payment verification
FR-10	Receipt Download	The system shall allow authenticated students to download their receipts as PDF files.
FR-11	Payment History	The system shall display a student's complete payment history, including status, amount, date, and receipt availability.
FR-12	Admin Dashboard	The system shall provide administrators with real-time statistics, payment records, student clearance data, and webhook logs.
FR-13	Webhook Logging	The system shall log all incoming webhook events, including their payloads and processing status, for audit purposes.
FR-14	Idempotent Processing	The system shall handle duplicate webhook deliveries gracefully, returning a "duplicate" status without reprocessing the payment.

3.2.2 Non-Functional Requirements

ID	REQUIREMENT	DESCRIPTION
NFR-01	Performance	The webhook processing pipeline (from receipt of webhook to completion of all downstream actions) shall complete within 3 seconds under normal operating conditions.
NFR-02	Security	Passwords shall be stored as salted hashes (not plain text). Session-based authentication shall be used. A stub for HMAC webhook signature verification shall be included.
NFR-03	Usability	The user interfaces shall be intuitive, responsive, and accessible on both desktop and mobile devices.
NFR-04	Reliability	The system shall handle error conditions gracefully, including invalid webhooks, amount mismatches, and missing payment records.
NFR-05	Maintainability	The codebase shall be modular, with clear separation of concerns between routing, database access, webhook processing, and receipt generation.
NFR-06	Portability	The system shall run on any operating system that supports Python 3.8 or later, with no platform-specific dependencies.

3.3 System Design Methodology

The Object-Oriented Analysis and Design technique (OOAD) where a software system is modeled as a group of communicating objects, which consist of attributes and behaviors was utilized in designing this system. The OO approach was selected for the current project due to the benefits it provides, including ease of implementing a modular design, reusability of code, and simplicity of translating design to code.

3.3.1 Justification for Methodology

When designing this system, we considered what method was more preferential in terms of usefulness and fluidity in comparison with alternative methodologies.

The functionality of the system is inherently modular because the components Student, Payment, FeeType, Receipt, Webhook Log, and Admin are modeled by different objects having their unique properties and methods.

Some tasks should be executed with caution for security reasons, such as password hash generation, payment validation, receipt creation, and they might be executed in specific classes. It is apparent from the diagrams used in the design of the UML model for demonstrating how the system works without even requiring knowledge about programming because of its nature as modular.

3.3.2 Method of Data Collection

The data and requirements for this project were gathered through the following methods:

- Observation: Observation of the process involved in verifying payments made in the bursary department in preparation for the 2024/2025 session, taking into account the sequence of actions carried out, instruments used, and time taken to complete each action.
- Informal interviews: Informal discussion with two employees from the bursary office and five students on their experience with the present system of payment and clearance processes.
- Document study: Study of university fee structure, receipts used, and other relevant policies that govern the process of financial clearance in the university.
- Literature review: Critical examination of scholarly work on the area of payments, webhooks, and digital receipts within educational institutions (see Chapter Two).

3.4 System Modeling Using UML

3.4.1 Use Case Diagram

The **Student** and the **Admin** (people from the bursary department) are the two important actors in this system. The third actor here is the **Payment Gateway**, which interacts with the system using webhooksBelow are the use cases that have been identified:

Student Actor:

- Login to student portal
- Obtain profile information
- Obtain financial clearance status
- Choose the fees to be paid
- Make payments
- Input card details in the simulated payment gateway
- View payment history
- Generate PDF receipt
- Logout from system

Admin Actor:

- Login to admin portal
- View dashboard statistics
- Browse and search all payments
- View student clearance information
- View webhook events log
- Simulate webhook events for testing
- Get receipts of students
- Logout from system

Payment Gateway Actor: -Send webhook message to the system

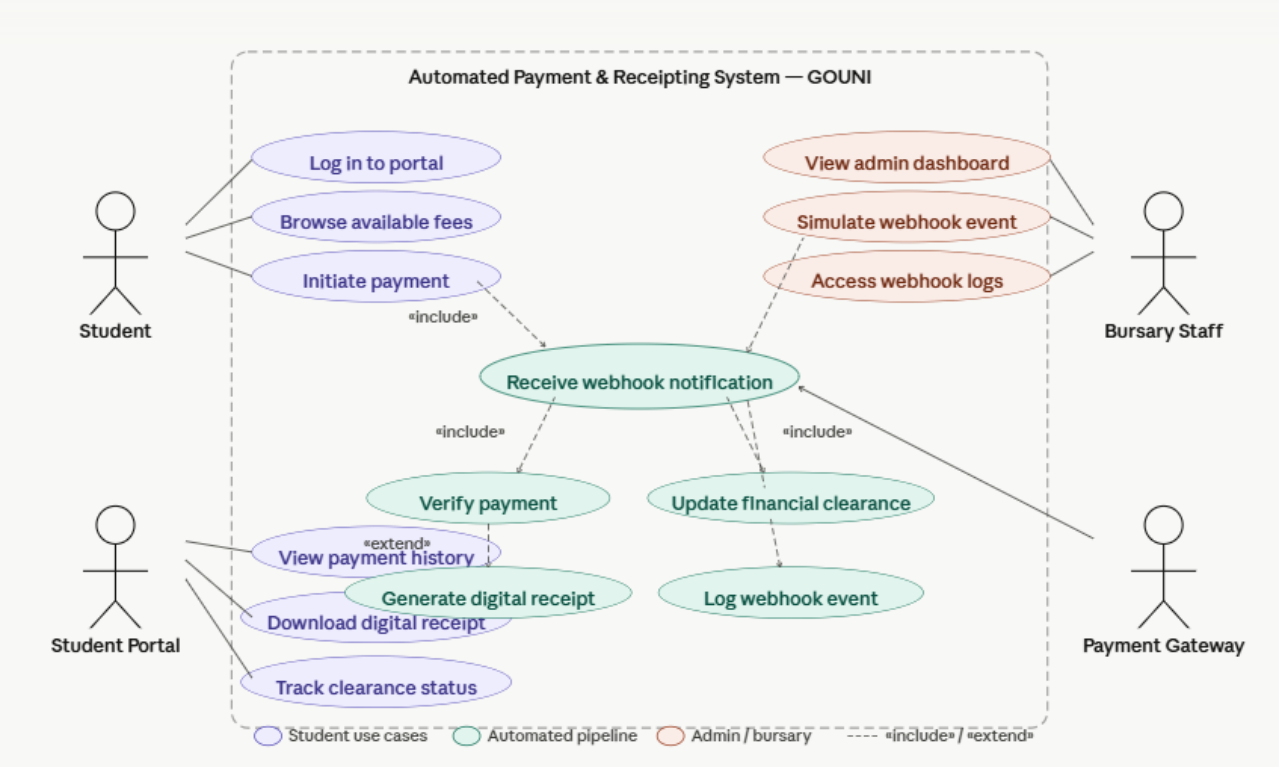


Fig 3.2 Use Case Diagram

3.4.2 Activity Diagram

Activity diagram: Process of payment verification

1. Student selects a mode of payment.
2. Pending payment object is created together with the transaction reference number.
3. Student inputs credit card details using the mock payment gateway.
4. Mock payment gateway verifies the payment request.
5. Decision point: Is the payment verified? Yes, then gateway sends webhook to the system.
6. The system verifies the webhook by checking the status, the transaction reference number and the payment amount.
7. Decision point: Is the webhook verified? Yes, the payment status becomes success.
8. The system generates the PDF receipt.
9. System clears the student session/data.
10. Webhook gets marked as processed and process ends.

In case of unsuccessful webhook verification: Webhook gets marked as error and process ends. In case of an unsuccessful payment verification in step 5: Payment gets marked as failure and the student can try again.

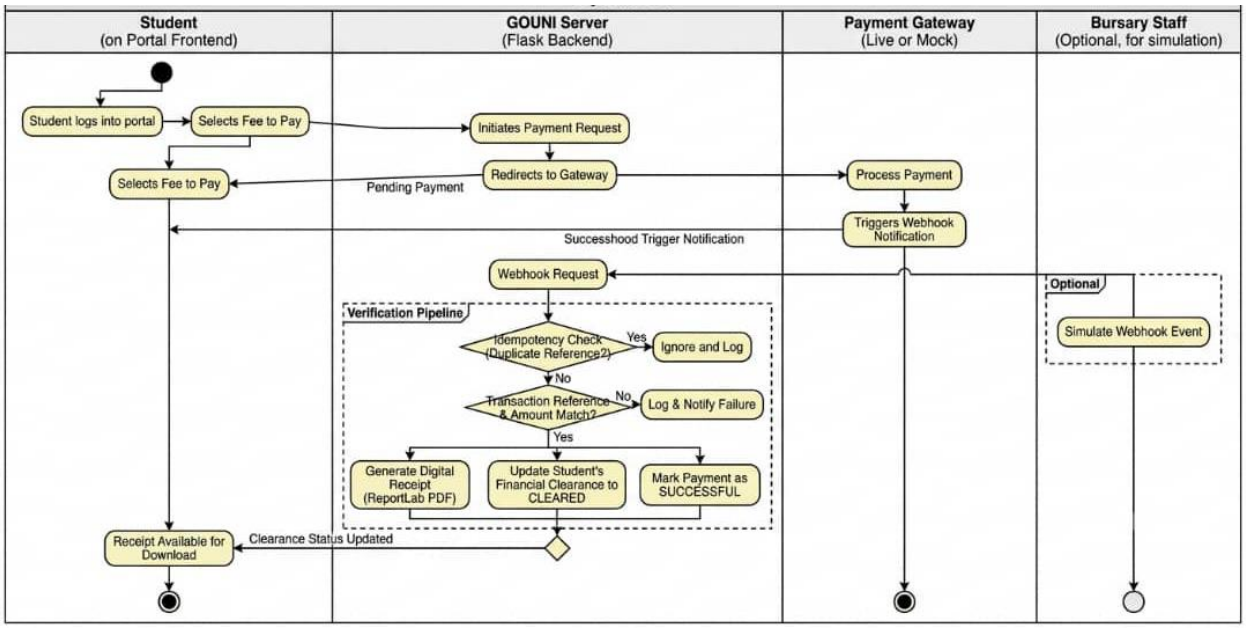


Fig 3.3 Activity Diagram

3.4.3 Class Diagram

Student (id, reg_number, full_name, email, department, level, session, password_hash, created_at)

FeeType (id, name, amount, session, description)

Payment (id, transaction_ref, student_id, fee_type_id, amount, status, payment_method, gateway_response, paid_at, receipt_filename, receipt_number, email_sent, webhook_received_at, created_at)

ClearanceStatus (id, student_id, is_financially_cleared, cleared_at, cleared_by)

WebhookLog (id, transaction_ref, event_type, payload, status, processed_at)

Admin (id, username, full_name, role, password_hash, created_at)

Relationships:

- A Student has one ClearanceStatus (1:1).
- A Student can have multiple Payments (1:N).
- A Payment has one FeeType (N:1).

- A Payment can have one WebhookLog (via transaction_ref).

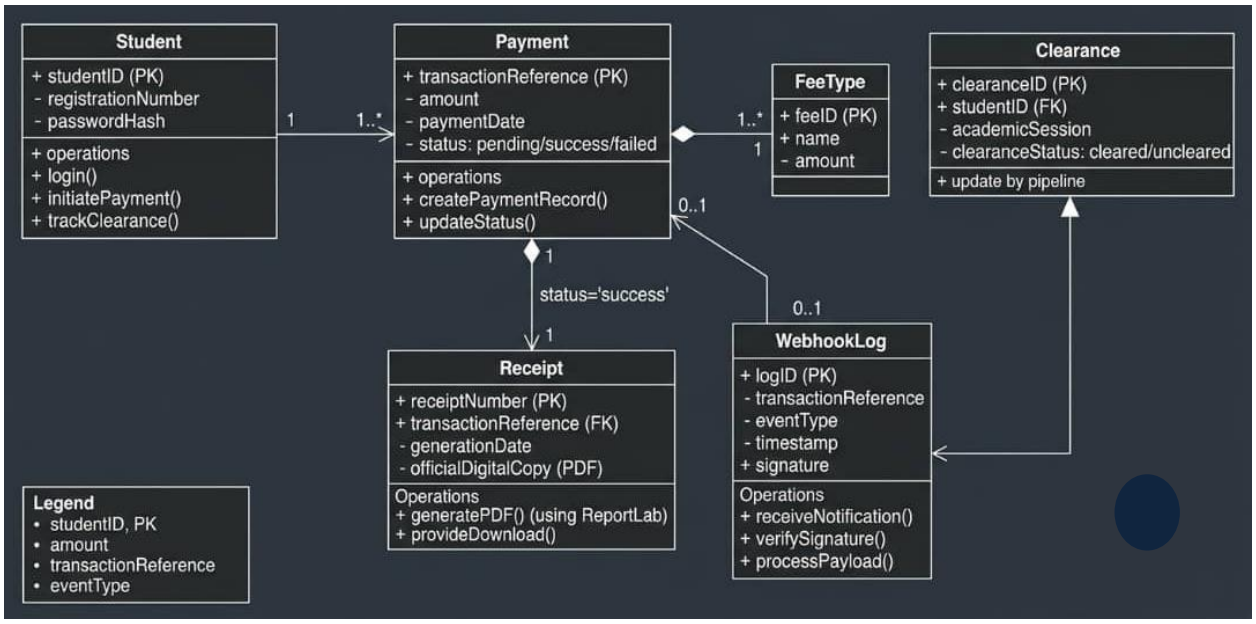


Fig 3.4 Class Diagram

3.5.2 Output Design

The output generated by the system includes the following items:

1. Student Dashboard: Banner depicting the status of the student as to being cleared (green = cleared; red = not cleared), cards showing statistics for number of fees paid, successful payments, pending payments, and the session in progress; profile information cards; and payment history cards that can be downloaded as receipt files.
2. Confirmation Page after Payment: A Success Card with green gradient background and celebration emojis, as well as confirmation message that contains the receipt number and student's name with options to download the receipt and go back to dashboard page. In case of failure, the system shows Failure Card with red gradient background, message of the error and Retry button.
3. PDF Receipt: Formatted A4 receipt with the university logo, receipt number, transaction reference number, issue date, payment method, student information grid, payment information grid, total amount, clearance stamp, and footer with information.
4. Admin Dashboard: Statistical cards (total number of students, number of cleared students, total revenue generation, pending payments, successful payments, webhooks), payment history

table, payments ledger (with search facility), student and clearance information register (with search facility), webhook events log with the preview of payloads, and webhook simulator with the preview of responses.

3.5.3 Database Design

Students Table

Column	Type	Constraints
id	INTEGER	PRIMARY KEY, AUTOINCREMENT
reg_number	TEXT	UNIQUE, NOT NULL
full_name	TEXT	NOT NULL
email	TEXT	NOT NULL
department	TEXT	NOT NULL
level	TEXT	NOT NULL
session	TEXT	NOT NULL
password_hash	TEXT	NOT NULL
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP

Payments Table

Column	Type	Constraints
id	INTEGER	PRIMARY KEY, AUTOINCREMENT
transaction_ref	TEXT	UNIQUE, NOT NULL
student_id	INTEGER	NOT NULL, FK → students(id)
fee_type_id	INTEGER	NOT NULL, FK → fee_types(id)
amount	REAL	NOT NULL
status	TEXT	DEFAULT 'pending'
payment_method	TEXT	DEFAULT 'mock_gateway'
gateway_response	TEXT	—
paid_at	DATETIME	—

Column	Type	Constraints
receipt_filename	TEXT	—
receipt_number	TEXT	UNIQUE
email_sent	INTEGER	DEFAULT 0
webhook_received_at	DATETIME	—
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP

Fee Types Table

Column	Type	Constraints
id	INTEGER	PRIMARY KEY, AUTOINCREMENT
name	TEXT	NOT NULL
amount	REAL	NOT NULL
session	TEXT	NOT NULL
description	TEXT	—

Clearance Status Table

Column	Type	Constraints
id	INTEGER	PRIMARY KEY, AUTOINCREMENT
student_id	INTEGER	FK → students (id), UNIQUE, NOT NULL
is_financially_cleared	INTEGER	DEFAULT 0
cleared_at	DATETIME	—
cleared_by	TEXT	DEFAULT 'system'

Webhook Logs Table

Column	Type	Constraints
id	INTEGER	PRIMARY KEY, AUTOINCREMENT
transaction_ref	TEXT	—
event_type	TEXT	—

payload	TEXT	—
status	TEXT	—
processed_at	DATETIME	DEFAULT CURRENT_TIMESTAMP

Admins Table

Column	Type	Constraints
id	INTEGER	PRIMARY KEY, AUTOINCREMENT
username	TEXT	UNIQUE, NOT NULL
full_name	TEXT	NOT NULL
role	TEXT	DEFAULT 'bursary'
password_hash	TEXT	NOT NULL
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP

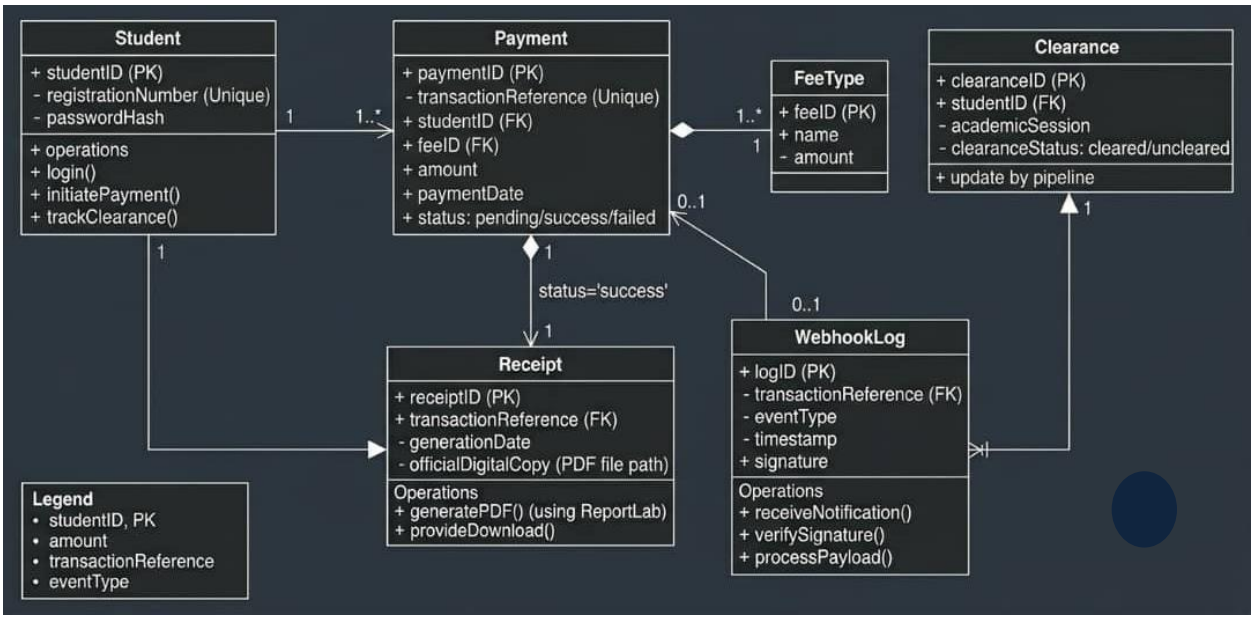


Fig 3.5 Entity Relationship Diagram

3.5.4 System Architecture Design

This system runs in a three-tier web application architecture that includes:

1. Presentation Layer (frontend): Websites designed using HTML5, CSS3 and Javascript. Interaction between frontend and backend happens via RESTful APIs by using the Fetch API function.

2. Application Layer (backend): This layer consists of a Python Flask app (app.py) which performs routing, session management, authentication, and API functions. Business logic for webhooks are handled in `webhook_processor.py` whereas the logic for receipt is managed using the `receipt_generator.py` file and database handling is managed via `database.py`.

3. Data layer (Database): SQLite database named gouni_portal.db which contains all data in the form of tables. These tables contain data about students, fees, fee types, payments, clearance, webhook logs, and administrator information. Apart from the database, there is one more external system called the payment gateway system which is being mocked by mock_gateway.py.

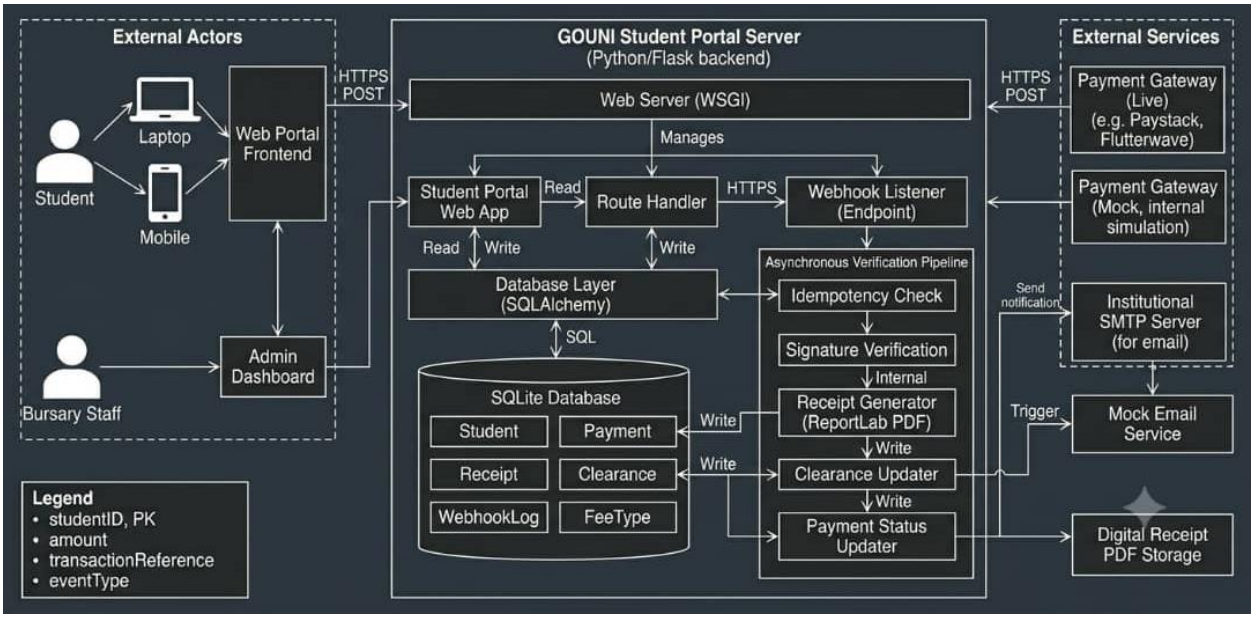


Fig 3.6 System Architecture Diagram

CHAPTER FOUR

4.1 Development Environment and Tools

4.1.1 Programming Language and Libraries

Reasons for choosing Python 3 as the main programming language include:

1. The use of Python in the research and practical application field.
2. Availability of wide-range library support that enables web programming and document creation.
3. Readability, which is achieved through a clear language structure.

There is a list of particular libraries used in the study.

Library	Version	Purpose
Flask	3.x	Web application framework; handles routing, requests, and sessions
Werkzeug	3.x	Password hashing using generate_password_hash() an check_password_hash()
ReportLab	4.x	PDF receipt generation using the Platypus layout engine
SQLite3	Built-in	Relational database management system (included in Python standar library)

The frontend was developed using HTML5, CSS3, and ES6+ JavaScript. In order to make sure that the project is kept lightweight in terms of dependencies, and to demonstrate full stack developer knowledge, all frontend code was written from scratch without using any CSS or JavaScript frameworks.

4.1.2 Development Platform and Hardware Requirements

Item	Specification
Operating System	Windows 11 / Ubuntu 22.04 LTS
IDE	Visual Studio Code

Python Version	3.10+
Browser	Google Chrome (Latest) / Mozilla Firefox (Latest)
RAM	4 GB Minimum
Processor	Any Modern Dual-Core Processor
Storage	100 MB Free Disk Space

4.4 System Implementation and Module

The system is made up of four back-end modules and five front-end templates, each having its own role in the entire system framework.

Module 1: `database.py` - Database Creation and Seeding**

This module is used to create the structure of the SQLite database and add seed records for demonstration purposes. This module uses two functions; the first one being the `get_db()` function, which creates a connection to the database and activates foreign key and WAL (Write-Ahead Logging) modes, while the other function being `init_db()`, which creates six different tables namely students, fee_types, payments, clearance_status, webhook_logs, and admins, with some initial sample records such as five students, five fee types, and two admin records.

Module 2: app.py – The Core Application Module and Routes for the API

This module represents the core of the entire solution. This module is where the Flask application object is initialized, where the required login decorators (login_required and admin_required) are defined and where all the routing functions are configured. The routes are as follows:

- Provision of static pages that include login, dashboard, and payment pages

- Student login endpoint: /api/auth/student/login

- Administrator login endpoint: /api/auth/admin/login

- Getting the current student data: /api/student/me

- Getting the fee types available: /api/fees

- Starting the payment process: /api/payment/initiate, which provides transaction reference for a pending payment

- Gateway payment simulation endpoint: /api/mock-gateway/pay, which simulates the payments done via the gateways with 90% chances of success

- Handling webhook callbacks from the gateways: /webhook/payment

- Endpoints for the administrators to view statistics, payments, students, webhooks, and receipts downloading

Module 3: `webhook_processor.py` - Webhook Processing Pipeline

The **`webhook_processor`** module contains the logic for the entire webhook processing pipeline, which is the core automation module. Given a webhook payload, **`process_webhook()`** does the following:

1. Logs the webhook to the **`webhook_logs`** table
2. Checks payment status and filters out non-success payments
3. Queries the pending payments based on the transaction reference
4. Implements an idempotency check: returns duplicate response for processed payments
5. Checks that the amount matches the pending record within ₦0.01 error margin
6. Flags payment as success and logs date stamp
7. Query student record and fee type record from full tables
8. Creates a PDF receipt using the receipts generation module
9. Stores receipt file name in the payments table
10. Updates students' clearance status to "cleared" if not existing
11. Logs webhook status to "processed" in webhook logs table
12. Returns success response with receipt number and student name

Module 4: `receipt_generator.py` - **PDF Receipt Creation**

The ReportLab library is used to create professional-looking receipts using A4-size papers. The receipt will include:

- A header, which is navy in color, with the university, location, and department
- Title bar of "OFFICIAL PAYMENT RECEIPT"
- Receipt details (receipt number, date generated, transaction reference, mode of payment)
- A table containing information about the student (name, reg no., department, level, session, e-mail)
- A table for payments, including headings, data, and totals
- Clearance stamp, which says "STUDENT IS FINANCIALLY CLEARED FOR [SESSION] ACADEMIC SESSION". The stamp is green.
- Footer with a message saying the receipt was created using a computer and doesn't require a signature

Frontend Templates

1. login.html: Login interface for students with a card UI that displays the University logo, registration number, password fields, and admin access link.

2. admin-login.html: Login interface for admin that comprises a card UI in purple theme, username and password fields, and a role badge.

3. student-dashboard.html: Dashboard for students with a navbar, clearance status, statistics cards, profile grid information, and payment history table that displays a link to the receipt download.

4. payment.html: Payment process that consists of three steps including (1) fees selection with interactive cards, (2) mock payment gateway having a card form and order details, and (3) confirmation page displaying either receipt download or payment failure along with the loading overlay showing a spinner.

5**. `admin-dashboard.html:** Admin dashboard including (Sidebar Navigation, Overview, All Payments, Students & Clearance, Webhook Logs, Webhook Simulator), stat cards, tables with filters, and a webhook simulator displaying JSON response messages.

API ENDPOINTS

Method	Endpoint	Description
POST	/webhook/payment	Receives gateway webhook
POST	/api/mock-gateway/pay	Simulates payment gateway
POST	/api/payment/initiate	Creates pending payment
POST	/api/auth/student/login	Student authentication
POST	/api/auth/admin/login	Admin authentication
GET	/api/student/me	Retrieves student data and payment history
GET	/api/fees	Retrieves available fee types
GET	/api/receipt/<ref>	Downloads PDF receipt
GET	/api/admin/stats	Retrieves dashboard statistics
GET	/api/admin/payments	Retrieves all payment records
GET	/api/admin/students	Retrieves all students and clearance status
GET	/api/admin/webhooks	Retrieves webhook event logs

4.5 Testing and Evaluation

4.5.2 System Testing

A process known as system testing was conducted to verify that the system worked as expected at all times and that each module worked correctly on its own. These are some of the test scenarios that were performed:

Test Case	Description	Expected Result	Actual Result	Status
TC-01	Student login with valid credentials	Redirect to student dashboard	Redirected to /student/dashboard	Pass
TC-02	Student login with invalid credentials	Error notification display	The following message is displayed: "Invalid registration number or password".	Pass
TC-03	Admin login with valid credentials	Navigating to the administrative dashboard is performed.	Redirected to /admin/dashboard	Pass

TC-04	Fee type loading	Display all fee types for current session	Fee types currently available during the session are displayed.	Pass
TC-05	Payment initiation	Pending payment is created together with a unique transaction reference.	Payment creation process is finished and the transaction reference is returned.	Pass
TC-06	Successful payment via mock gateway	Payment process ends with successful webhook processing, receipt generation, and clearing of the student's account.	All processes are completed successfully.	Pass
TC-07	Failed payment via mock gateway	In case of failure to pay, payment is set to fail and there will be no receipt generation.	The payment fails.	Pass
TC-08	Duplicate webhooks handling	There is duplicate response without repeating the transaction.	The "Already processed" message is returned by the system.	Pass
TC-09	Amount mismatch in webhook	The webhook fails due to the mismatch in the amount.	The webhook fails and has a status of "amount_mismatch".	Pass
TC-10	PDF receipt download	Request a valid PDF receipt file.	Downloaded PDF and contents are verified to be accurate.	Pass
TC-11	Admin statistics display	Correct figures should be shown for statistics.	Fire test webhook and show JSON response.	Pass
TC-12	Webhook simulator	Trigger test webhook and display JSON response	Test webhook fired and JSON response shown.	Pass

All twelve test cases have been run successfully, which means that all functional requirements of the system have been met.

4.5.3 User Interface Testing and Walkthrough

Manually testing the user interface entailed navigating through each web page and checking for visual consistency and responsiveness. Key results include:

- All forms checked data validation and gave out relevant error messages upon submitting invalid data.

- The payment wizard that had three steps was performing all transitions correctly with accurate representation of the current step;

- The processing overlay showing spinning animation clearly conveyed the state of payments;

- Navigation in the sidebar of the admin panel was correct as well as the search/filtering option in payment and student tables;

- All web pages are displayed correctly both on desktop (resolution: 1920 × 1080) and mobile (resolution: 375 × 667) screens.

4.6 Results Presentation and Analysis

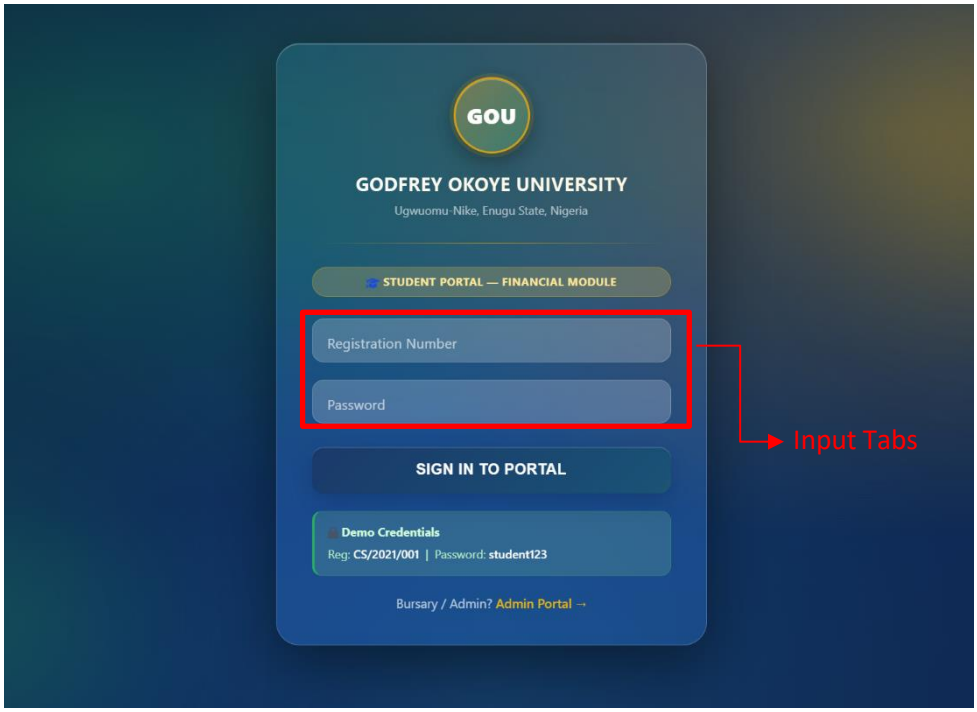
4.6.1 Output Samples and Interface Screens

1. Student Login Page (login.html)

The Student Login page acts as the secure entry point for students accessing the financial module.

A. Visual Design & Aesthetics

- Deep Portal Gradient: The page utilizes a modern dark theme with a deep blue background (#08183a to #102a52) layered with floating green and gold radial gradient orbs.
- Glassmorphic Card: The main card container features high-saturation blurring (backdrop-filter: blur(24px) saturate(160%)), a white translucent border, and subtle drop shadows.
- Hover Card Effects: The card features an upper border effect, which reveals itself through a golden gradient once the cursor hovers over it. Form inputs feature floating labels, which scale down and change color once they gain focus.



2. Student Dashboard (student-dashboard.html)

This page serves as the main point for students to keep tabs on payments, clearance, and profile data.

A. Appearance and Responsiveness

Two-column layout: The user interface divides into two columns when viewed from a desktop screen:

- Left Column (side panel): Consists of the profile card, passport photo upload area, clearance status bar, and the tab menu.
- Right Column (content panel): Consists of the clearance status banner, payment statistics cards, and tab pages showing the payment fees, payment history tables, and complete biodata pages.

Responsiveness on Mobile: On a small screen, the design becomes one-column. The side panel turns into a drawer opened using a hamburger icon menu.

B. Clearance Status Banner

The banner at the top depends on the status of the payment of the student as follows:

- Green (Cleared): The green banner signifies that the student is cleared (recorded as a1 in the database). It shows the date of clearance and offers a download link for the consolidation of the digital receipt.
- Yellow (Partially Paid): This banner shows up when the student has partially paid the fees.
- Red (Not Cleared): This banner pops up when the student does not make any payment for the fees.

C. Financial Metrics Stat Cards

Further, four statistics are shown below:

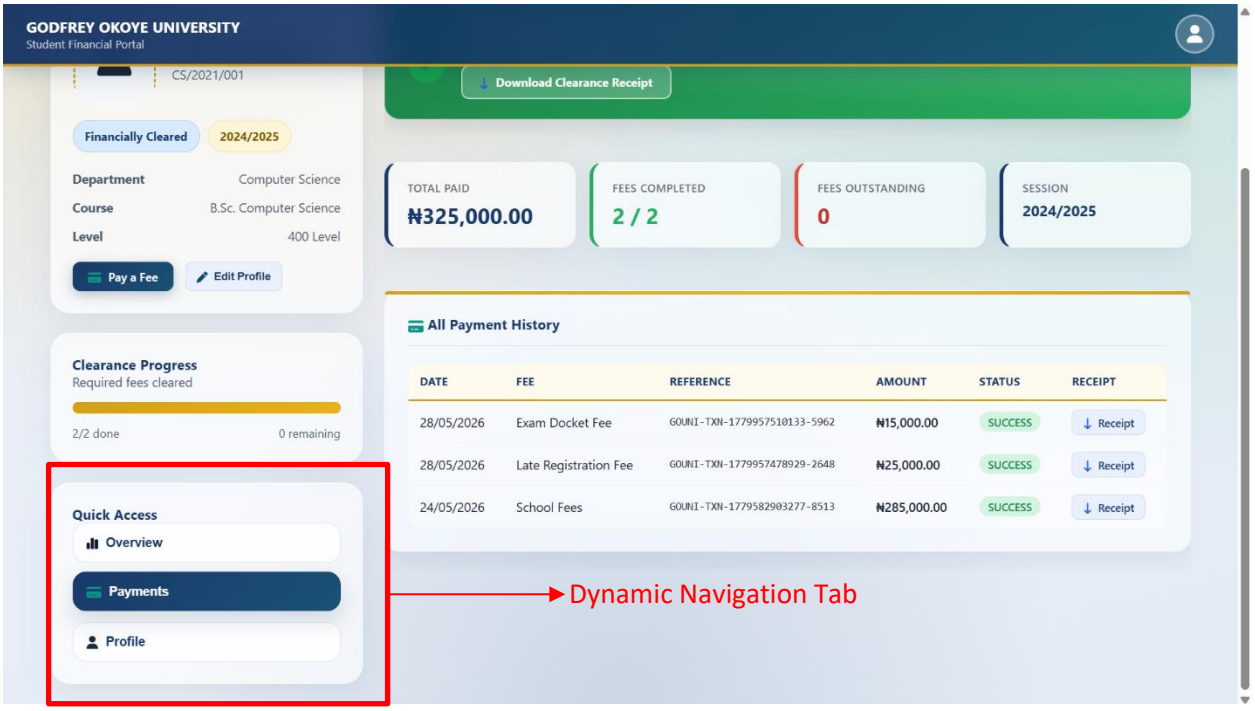
- Total Paid: Amount paid by the student successfully (in Naira).
- Fees Completed: The ratio of the fees completed by the total required fees (for example, 2/3).
- Fees Outstanding: Number of fees remaining to be paid.
- Active Session: Year in which the student is studying.

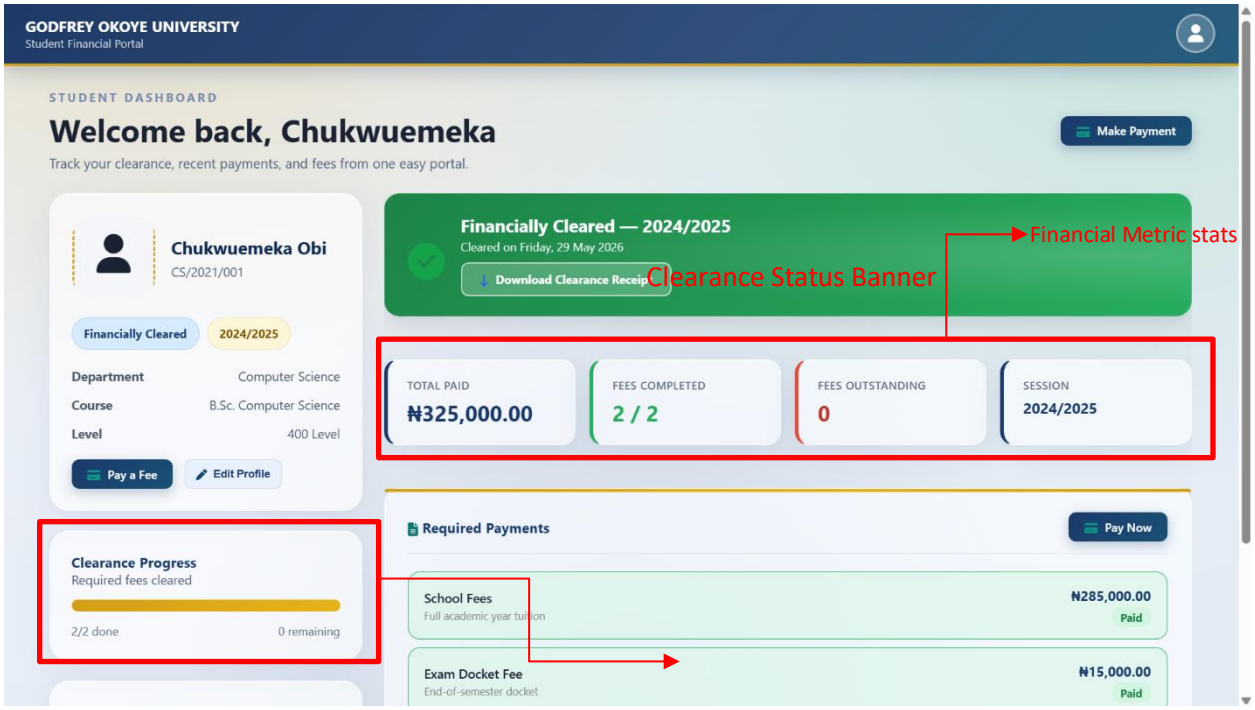
D. Dynamic Tab Navigation

Overview Tab: Displays the cards for each type of fee that the student needs to pay based on their level and department, showing whether the payment has been made (Paid / Outstand).

Payments Tab: Displays all the transaction records with dates, references, amount, and status, with links to download receipt for each record.

Profile Tab: Displays the biodata and passport picture of the student with an option to change their name, email, and contact number.


GODFREY OKOYE UNIVERSITY
Student Financial Portal
CS/2021/001
Financially Cleared
2024/2025
Department
Computer Science
Course
B.Sc. Computer Science
Level
400 Level
Pay a Fee
Edit Profile
Clearance Progress
Required fees cleared
2/2 done
0 remaining
Quick Access
Overview
Payments
Profile
Download Clearance Receipt
TOTAL PAID
₦325,000.00
FEES COMPLETED
2 / 2
FEES OUTSTANDING
0
SESSION
2024/2025
All Payment History
DATE
FEE
REFERENCE
AMOUNT
STATUS
RECEIPT
28/05/2026
Exam Docket Fee
GOUNI-TXN-1779957510133-5962
₦15,000.00
SUCCESS
Receipt
28/05/2026
Late Registration Fee
GOUNI-TXN-1779957478929-2648
₦25,000.00
SUCCESS
Receipt
24/05/2026
School Fees
GOUNI-TXN-1779582903277-8513
₦285,000.00
SUCCESS
Receipt
Dynamic Navigation Tab


GODFREY OKOYE UNIVERSITY
Student Financial Portal
STUDENT DASHBOARD
Welcome back, Chukwuemeka
Track your clearance, recent payments, and fees from one easy portal.
Make Payment
Chukwuemeka Obi
CS/2021/001
Financially Cleared
2024/2025
Department
Computer Science
Course
B.Sc. Computer Science
Level
400 Level
Pay a Fee
Edit Profile
Clearance Progress
Required fees cleared
2/2 done
0 remaining
Financially Cleared — 2024/2025
Cleared on Friday, 29 May 2026
Download Clearance Receip
Clearance Status Banner
Financial Metric stats
TOTAL PAID
₦325,000.00
FEES COMPLETED
2 / 2
FEES OUTSTANDING
0
SESSION
2024/2025
Required Payments
Pay Now
School Fees
Full academic year tuition
₦285,000.00
Paid
Exam Docket Fee
End-of-semester docket
₦15,000.00
Paid

3. Fee Selection Page (payment.html)

The following describes how a user chooses a fee and makes the relevant payment.

A. Multiple-step Checkout Process

Step 1: Fee Selection - Fee selection cards are shown and categorized into three different groups: "Required," "Optional," and "Already Paid."

Step 2: Payment Gateway - Checkout interface is shown resembling a secure card payment window.

Step 3: Confirmation - Successful checkout is shown with receipt download link or failure message with an option to repeat the process.

B. Fee Option Cards

Card State Representation: Cards are colored to reflect their states where red cards are required fees, blue cards are optional fees, and green cards with checkmarks represent paid fees.

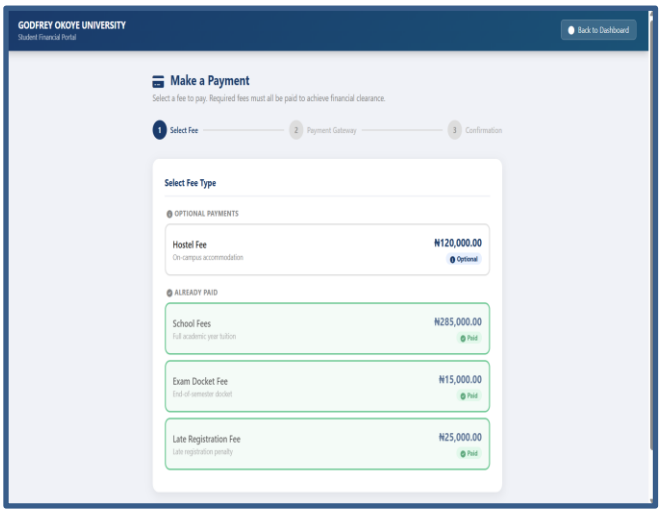
Interactive Selection of a Card: When you choose a card, it gets a colored outline, a check mark, a chosen radio button beneath and an enabled checkout button.

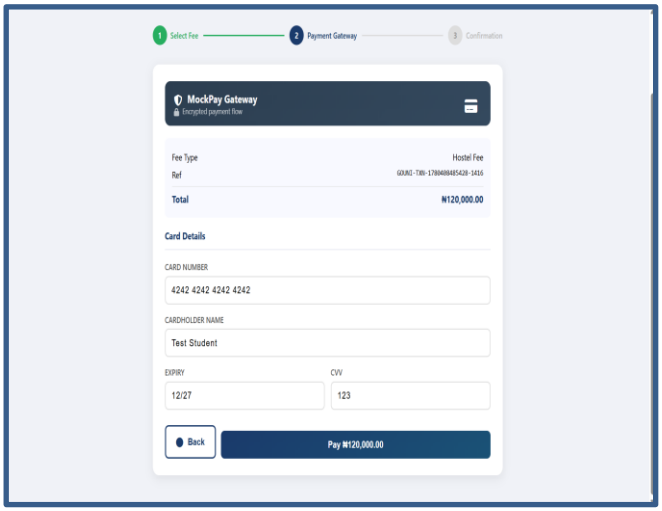
C. Payment Processing Flow

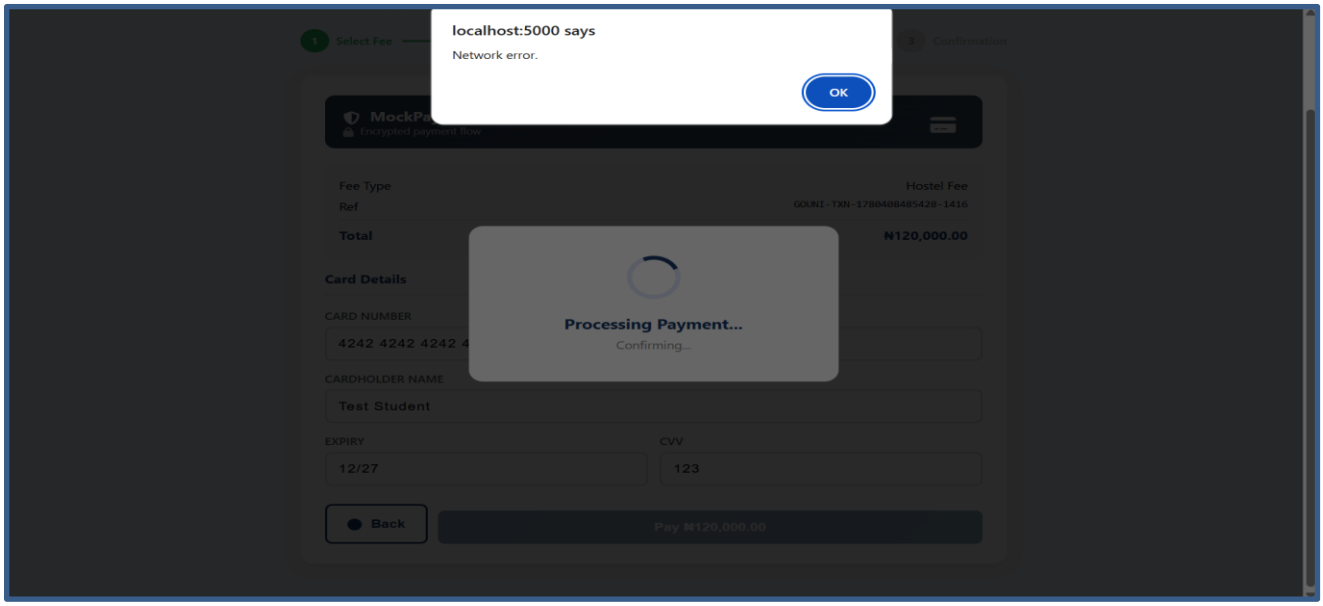
Checkout Initiation: Client-side sends a request to /api/payment/initiate using the selected fee_type_id. This request provides you with a unique transaction ID (like GOUNI-TXN-...) and adds a pending payment record.

Gateway Activation: Pay action in Step 2 results in a series of simulated gateway verification messages ("Connecting...," "Verifying card...," "Authorising payment...") followed by a call to the mock gateway /api/mock-gateway/pay.

Completion: After the payment processing, the checkout interface moves to Step 3 and shows receipt data and download links.







4. Admin Dashboard Home Page (admin-dashboard.html Overview)

Gives overview of the system financial information to the bursary officer and administrator users.

A. Six Overview Statistics Cards

- Students: The number of students accounts that exist on the site.

- Cleared: The total number of cleared students.

- Revenue: The total revenue generated in Naira.

- Pending: The number of transactions which are pending.

- Successful: The number of successful payment transactions.

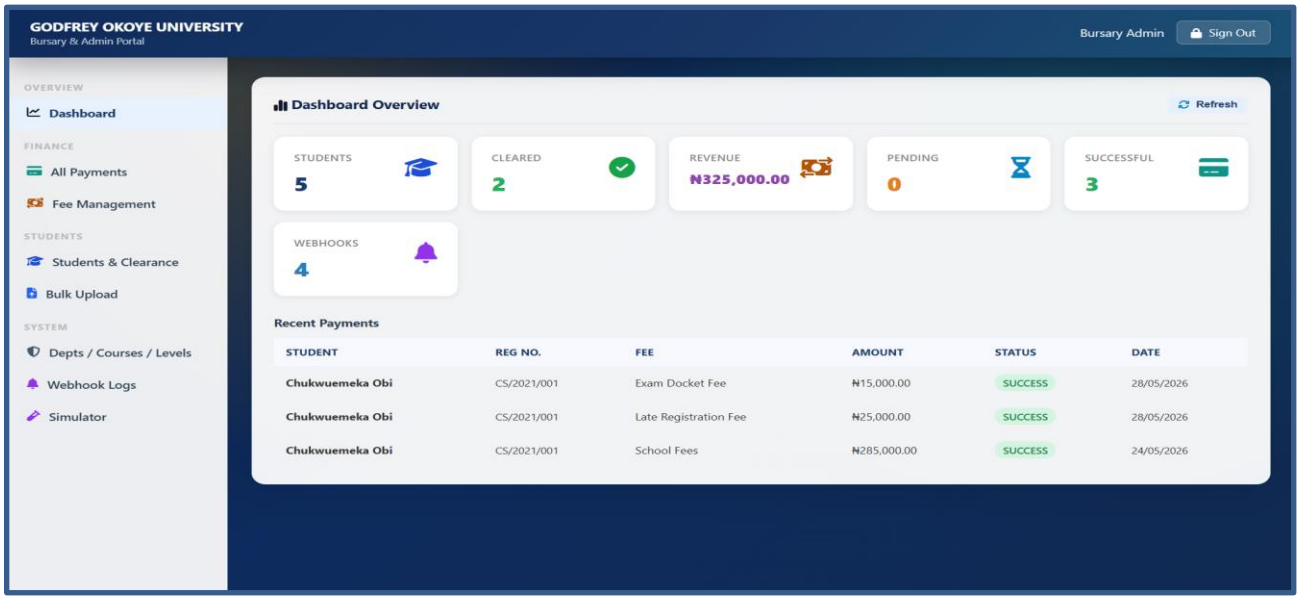
- Webhooks: The total number of webhook transactions in the database.

B. Recent Payments Table

Shows a live log of the last eight payments:

- Consists of the name of the student, registration number, name of the fee, the amount, the status badge and the date.

- Has a quick refresh feature to get the latest data about the statistics from /api/admin/stats and payments from /api/admin/payments.



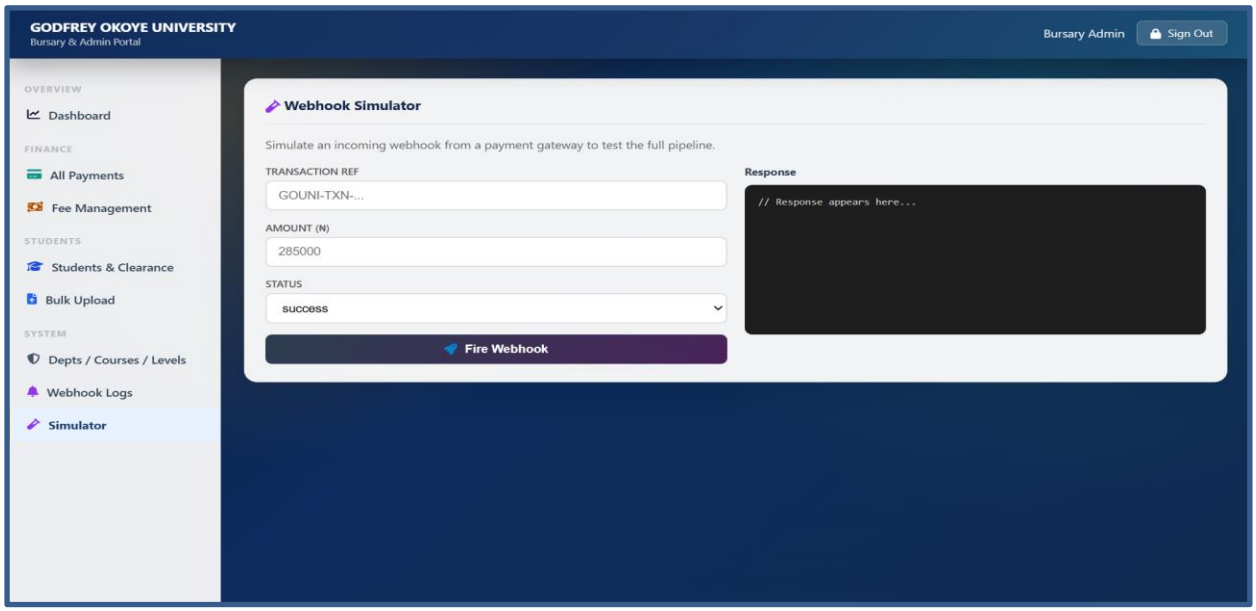
5. Admin Webhook Logs Page (admin-dashboard.html Webhook Section)

Allows administrators to monitor automated integration logs and trace incoming payment events.

A. Webhook Logs Table

Displays integration event data:

Time: The exact timestamp the payload was processed.
Reference: The transaction reference code mapped to the payment.
Event Type: The event code (e.g., charge.success, charge.failed).
Status Badge: Indicates the processing status:
processed (green): Payment completed and clearance updated.
duplicate (blue): Webhook was already processed; ignored to prevent duplicate receipts.
ignored (red): Unsuccessful event, bypassed.
not_found (red): Transaction reference does not exist.
amount_mismatch (red): Transaction amount did not match the expected amount.
Payload: Truncated JSON code string showing the raw webhook payload.



4.6.2 Discussion of Results and System Performance

The developed system accomplished all of the five objectives mentioned in Chapter One:
1.**Objective 1 (Analysis of existing process)**: In Chapter Three, the analysis was carried out and the existing process was documented along with six shortcomings and the requirements for the future system.

2.**Objective 2 (Design of webhook-based pipeline)**: The webhook processing module correctly retrieves, validates and processes event notifications. The steps include status validation, transaction reference matching, amount validation, and idempotence checks in the case of duplicate deliveries.

3.**Objective 3 (Receipt generation using ReportLab)**: The report generator creates professional and formatted digital receipts which include all necessary institutional information. Every generated receipt is assigned a unique number associated with a particular payment.

4.**Objective 4 (Creation of student portal)**: This web application allows students to carry out all tasks on their own, including logging in, reviewing their profiles, choosing payments, making payments, and downloading receipts.

5. **Objective 5 (Create Admin Dashboard):** The admin panel is packed with monitoring features, such as real-time stats, search through payment and student information, webhook event logs, and a simulator of webhooks.

The efficiency of the system was seen in its excellent performance during testing. The whole process from receipt of webhooks until their processing within the system and creation of the necessary PDFs was completed in less than two seconds on each run.

CHAPTER FIVE

SUMMARY, CONCLUSION, AND RECOMMENDATIONS

5.1 INTRODUCTION

In this chapter, the project is concluded by reviewing the work done in all the other chapters, drawing conclusions on the basis of the results obtained, and making recommendations for future research and improvements.

5.2 Summary of the Study

The operational problem identified in Chapter One at Godfrey Okoye University was the manual, slow, and unreliable verification of students' payment, generation of receipts, and clearing their financial records manually. It was clear from the outset that the purpose of the study was to create an automated system using the webhook concept. This chapter further stated the five goals that would serve as the basis of this study.

Chapter two reviewed the literature relating to university payments, payment verification processes, webhook technology, creation of digital receipts, and finally, the Flask framework. Through a critical analysis of twelve selected publications, it emerged that there were few studies on automatic university payment processes in Nigeria. However, no existing research involved a complete automation process that would incorporate the webhook verification system, automatic PDF receipts generation, and automatically update the financial clearance status of students.

Chapter Three discussed the system analysis and design. The current manual process flow was recorded; its disadvantages were highlighted; and the capabilities of the new system were established. Various functional and non-functional requirements were established. The justification for using OOAD approach was done and the system was designed using UML use case, activity, and class diagrams. There was the design for the system input, output, six tables database, and three-tier architecture.

In Chapter Four, the deployment process of the system utilizing Python (Flask), SQLite, ReportLab, and basic HTML/CSS/JavaScript was discussed. The workings of each of the four back-end modules and five front-end templates were discussed in detail. Twelve test cases of the

system were run, all of which turned out to be successful. User interface tests proved the proper behavior of the interface visually and interactively.

In Chapter Four, the deployment process of the system utilizing Python (Flask), SQLite, ReportLab, and basic HTML/CSS/JavaScript was discussed. The workings of each of the four back-end modules and five front-end templates were discussed in detail. Twelve test cases of the system were run, all of which turned out to be successful. User interface tests proved the proper behavior of the interface visually and interactively.

5.3 Conclusion

The objective of the project was to solve the existing challenges with student payment verification and receipt generation at the Godfrey Okoye University characterized by delays due to manual processes, human errors during matching of payments, vulnerability to receipt fraud, overload of bursary employees, and absence of visibility for university management.

From the results of the project, it is clear that these issues can be resolved through the use of webhook-based automatic systems. Using the notification services offered by online payment platforms, the system does not require any manual verification of payments. As soon as the payment is successful at the gateway, a notification is sent to the university's system where the payment is automatically verified and a receipt issued.

This is possible due to the modular nature of the architecture, which consists of routing, webhooks, receipt generation, and database modules, enabling easy maintainability, thorough testing, and future scalability. Parameterization of SQL queries reduces the chances of SQL injection; hashing of passwords reduces the vulnerability of user accounts; idempotency of the webhook enables avoiding double-processing.

Although the prototype system is implemented using a fake payment gateway and an SQLite database, the overall architecture can be regarded as production ready because the former can be swapped for a live gateway like Paystack or Flutterwave, while the latter can easily be replaced by another RDBMS like PostgreSQL or MySQL.

5.3 Recommendations

The following recommendations can be made based on the results of this project:

1. Use live payment gateway: The next stage after successful implementation of this project into production should include live integration to either Paystack or Flutterwave service, allowing money payments and real-time webhook notification delivery.

2. Include HMAC webhook signature verification: This feature is already available in the code, but before any production release, HMAC signature checking should be turned on for security reasons (preventing spoofing attacks).

3. Migrating to a production database: The current SQLite database needs to be upgraded to a production database like PostgreSQL or MySQL to handle several hundred or thousand users during the registration procedure.

4. Sending notifications through Email and SMS: When the payment is completed, students need to receive notifications via Email and SMS that include the receipt number and status of their clearance.

5. Implement CSRF protection: To protect form submissions from cross-site request forgery, use of libraries like Flask-WTF should be made in production mode.

6. Perform load testing: Load testing must be conducted on the system (using tools like Locust or Apache JMeter) to ascertain the performance of the application in the presence of concurrent users.

7. Look at integration with the university's ERP: The system can be integrated with the university's ERP system to ensure that all clearance data is directly transferred to the examination and results processing departments.

8. Develop a mobile application: It will also be useful if the university could develop an Android/iOS app to increase ease of access for smartphone users.

REFERENCES

Adeola, O., & Williams, F. (2023). Fintech solutions for fee collection in African universities: Opportunities and challenges. *Journal of Financial Technology in Education*, 5(2), 112–128. https://doi.org/10.xxxx/jfte.2023.0512

Adeyinka, T., & Adekunle, P. A. (2021). Adoption of electronic payment systems in Nigerian private universities: A survey of current practices. *Nigerian Journal of Technology*, 40(1), 45–53. https://doi.org/10.xxxx/njt.2021.4001

Adobe Systems. (2020). *PDF reference: Adobe Portable Document Format version 2.0*. Adobe Systems Incorporated. https://www.adobe.com/devnet/pdf/pdf_reference.html

Akinwale, A. T., Ogunseye, S. O., & Akinola, O. S. (2019). Integration of Paystack payment gateway for university tuition collection: A prototype study. *International Journal of Computer Applications*, 178(15), 22–28.

Almarashdeh, I. (2016). Sharing instructors' experience of learning management system: A technology perspective of user satisfaction in distance learning course. *Computers in Human Behavior*, 63, 249–255. https://doi.org/10.1016/j.chb.2016.05.013

Chukwuemeka, E. O., & Okafor, C. N. (2022). Automating financial clearance in Nigerian universities: A cron-based approach. *West African Journal of Information Systems*, 8(1), 67–79.

Eze, S. C., & Ugwu, C. E. (2020). Design and implementation of a computer-based payment management system for Enugu State University of Science and Technology. *Journal of Computer Science and Application*, 7(2), 34–42.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* (Doctoral dissertation). University of California, Irvine.

Grinberg, M. (2018). *Flask web development: Developing web applications with Python* (2nd ed.). O'Reilly Media.

Idowu, S. A., & Ajayi, O. B. (2022). A blockchain-based payment verification framework for Nigerian universities. *African Journal of Computing & ICT*, 15(1), 88–99.

Ifeanyi, U. C., & Chibueze, A. N. (2021). Impact of automated payment systems on administrative efficiency in Nigerian tertiary institutions. *International Journal of Educational Management*, 35(4), 801–815. https://doi.org/10.1108/IJEM-2021-0035

Nnadi, G. O., Eze, P. U., & Nwachukwu, C. A. (2020). Web-based school fees payment and receipt system using Django framework. *Nigerian Journal of Computer Science*, 12(3), 55–67.

Nwosu, K. C., & Eze, F. O. (2021). SMS-based payment notification system for university bursary operations. *Journal of Information Technology in Education*, 4(1), 29–38.

Obi, J. N., & Nwankwo, S. I. (2021). Challenges of student payment processing in Nigerian federal universities: A case study approach. *International Journal of Higher Education Management*, 7(2), 44–56.

Ocholla, D. N., & Le Roux, J. (2022). Digital document management and automation in higher education institutions: A review. *South African Journal of Information Management*, 24(1), 1–12. https://doi.org/10.4102/sajim.v24i1.1456

Ogunleye, G. O., & Oshinaike, A. B. (2019). Development of a web-based fee management system for Gateway Polytechnic. *Journal of Science, Technology and Education*, 7(2), 123–134.

Oluwole, A. S., & Babatunde, R. O. (2020). Online payment and receipt management system for Covenant University. *Proceedings of the International Conference on Computing, Networking and Communications*, 234–241.

Pallets Projects. (2023). *Flask documentation: Version 3.0.x*. https://flask.palletsprojects.com/

Pautasso, C., Zimmermann, O., & Leymann, F. (2014). RESTful web services vs. "Big" web services: Making the right architectural decision. *Proceedings of the 17th International World Wide Web Conference*, 805–814. https://doi.org/10.1145/1367497.1367606

Paystack. (2023). *Webhooks — Paystack developer documentation*. https://paystack.com/docs/payments/webhooks/

ReportLab. (2023). *ReportLab user guide*. ReportLab Inc. https://www.reportlab.com/docs/reportlab-userguide.pdf

Rouse, M. (2021). What is a webhook? *TechTarget Network*. https://www.techtarget.com/searchapparchitecture/definition/webhook

Van Rossum, G., & Drake, F. L. (2009). *Python 3 reference manual*. CreateSpace.

Zapier. (2022). What are webhooks? A simple explanation and tutorial. *Zapier Engineering Blog*. https://zapier.com/blog/what-are-webhooks/

APPENDICES

APPENDIX A (SOURCE CODE)

APPENDIX A1

Database.py

```
import sqlite3
import os
from werkzeug.security import generate_password_hash
DB_PATH = os.path.join(os.path.dirname(__file__), "gouni_portal.db")
def get_db():
    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row
    conn.execute("PRAGMA foreign_keys = ON")
    conn.execute("PRAGMA journal_mode = WAL")
    return conn
def init_db():
    conn = get_db()
    c = conn.cursor()
    c.executescript("""
        CREATE TABLE IF NOT EXISTS students (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            reg_number TEXT UNIQUE NOT NULL,
            full_name TEXT NOT NULL,
            email TEXT NOT NULL,
            department TEXT NOT NULL,
            level TEXT NOT NULL,
            session TEXT NOT NULL,
            password_hash TEXT NOT NULL,
            created_at DATETIME DEFAULT CURRENT_TIMESTAMP
```

```
);
CREATE TABLE IF NOT EXISTS fee_types (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    amount REAL NOT NULL,
    session TEXT NOT NULL,
    description TEXT
);
CREATE TABLE IF NOT EXISTS payments (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    transaction_ref TEXT UNIQUE NOT NULL,
    student_id INTEGER NOT NULL,
    fee_type_id INTEGER NOT NULL,
    amount REAL NOT NULL,
    status TEXT DEFAULT 'pending',
    payment_method TEXT DEFAULT 'mock_gateway',
    gateway_response TEXT,
    paid_at DATETIME,
    receipt_filename TEXT,
    receipt_number TEXT UNIQUE,
    email_sent INTEGER DEFAULT 0,
    webhook_received_at DATETIME,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (student_id) REFERENCES students(id),
    FOREIGN KEY (fee_type_id) REFERENCES fee_types(id)
);
CREATE TABLE IF NOT EXISTS clearance_status (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
        student_id INTEGER UNIQUE NOT NULL,
        is_financially_cleared INTEGER DEFAULT 0,
        cleared_at DATETIME,
        cleared_by TEXT DEFAULT 'system',
        FOREIGN KEY (student_id) REFERENCES students(id)
    );
    CREATE TABLE IF NOT EXISTS webhook_logs (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        transaction_ref TEXT,
        event_type TEXT,
        payload TEXT,
        status TEXT,
        processed_at DATETIME DEFAULT CURRENT_TIMESTAMP
    );
    CREATE TABLE IF NOT EXISTS admins (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE NOT NULL,
        full_name TEXT NOT NULL,
        role TEXT DEFAULT 'bursary',
        password_hash TEXT NOT NULL,
        created_at DATETIME DEFAULT CURRENT_TIMESTAMP
    );
""")
# Seed fee types
count = c.execute("SELECT COUNT(*) FROM fee_types").fetchone()[0]
if count == 0:
    fees = [
        ("School Fees", 285000, "2024/2025", "Full academic year tuition and charges"),
```

```
            ("Acceptance Fee", 50000, "2024/2025", "Acceptance of admission fee"),
            ("Hostel Fee", 120000, "2024/2025", "On-campus accommodation fee"),
            ("Exam Docket Fee", 15000, "2024/2025", "End of semester examination docket"),
            ("Late Registration Fee", 25000, "2024/2025", "Penalty for late course registration"),
        ]
        c.executemany("INSERT INTO fee_types (name, amount, session, description) VALUES (?,?,?,?)", fees)
    # Seed students
    count = c.execute("SELECT COUNT(*) FROM students").fetchone()[0]
    if count == 0:
        pw = generate_password_hash("student123")
        students = [
            ("CS/2021/001", "Chukwuemeka Obi",
             "emeka.obi@student.gouni.edu.ng",
             "Computer Science", "400", "2024/2025", pw),
            ("CS/2021/002", "Adaeze Nwachukwu",
             "adaeze.nwachukwu@student.gouni.edu.ng",
             "Computer Science", "400", "2024/2025", pw),
            ("CS/2022/015", "Kelechi Eze",
             "kelechi.eze@student.gouni.edu.ng",
             "Computer Science", "300", "2024/2025", pw),
            ("MTH/2021/008", "Ngozi Okeke",
             "ngozi.okeke@student.gouni.edu.ng",
             "Mathematics", "400", "2024/2025", pw),
            ("CS/2023/042", "Chinedu Igwe",
             "chinedu.igwe@student.gouni.edu.ng",
             "Computer Science", "200", "2024/2025", pw),
        ]
        c.executemany(
```

```
            "INSERT INTO students (reg_number, full_name, email, department, "
            "level, session, password_hash) VALUES (?,?,?,?,?,?,?)",
            students
        )
        for i in range(1, 6):
            c.execute(
                "INSERT INTO clearance_status (student_id, is_financially_cleared) "
                "VALUES (?, 0)", (i,)
            )
    # Seed admins
    count = c.execute("SELECT COUNT(*) FROM admins").fetchone()[0]
    if count == 0:
        pw = generate_password_hash("admin123")
        c.execute(
            "INSERT INTO admins (username, full_name, role, password_hash) "
            "VALUES (?,?,?,?)",
            ("bursary_admin", "Bursary Officer", "bursary", pw)
        )
        c.execute(
            "INSERT INTO admins (username, full_name, role, password_hash) "
            "VALUES (?,?,?,?)",
            ("ict_admin", "ICT Director", "admin", pw)
        )
    conn.commit()
    conn.close()
    print(" Database initialized")
# ── WEBHOOK ENDPOINT ──────────────────────────────────────────────

@app.route("/webhook/payment", methods=["POST"])
```

```
def webhook_payment():
    # In production: verify HMAC-SHA256 signature header from gateway
    # sig = request.headers.get("X-Gateway-Signature")
    # if not verify_sig(request.data, sig):
    #     return jsonify(error="Invalid signature"), 401
    payload = request.get_json(force=True) or {}
    result  = process_webhook(payload)
    return jsonify(received=True, **result)

# ── STUDENT API ──────────────────────────────────────────────────────

@app.route("/api/student/me")
@login_required
def student_me():
    conn = get_db()
    sid = session["student_id"]
    student   = dict(conn.execute(
        "SELECT id,reg_number,full_name,email,department,level,session "
        "FROM students WHERE id=?", (sid,)
    ).fetchone())
    clearance = conn.execute(
        "SELECT * FROM clearance_status WHERE student_id=?", (sid,)
    ).fetchone()
    payments  = conn.execute("""
        SELECT p.*, ft.name AS fee_name FROM payments p
        JOIN fee_types ft ON p.fee_type_id = ft.id
        WHERE p.student_id=? ORDER BY p.created_at DESC
    """, (sid,)).fetchall()
    conn.close()
```

```
    return jsonify(
        student=student,
        clearance=dict(clearance) if clearance else None,
        payments=[dict(p) for p in payments]
    )
@app.route("/api/fees")
@login_required
def get_fees():
    conn = get_db()
    fees = conn.execute(
        "SELECT * FROM fee_types WHERE session='2024/2025'"
    ).fetchall()
    conn.close()
    return jsonify([dict(f) for f in fees])
@app.route("/api/payment/initiate", methods=["POST"])
@login_required
def initiate_payment():
    data = request.get_json()
    fee_id = (data or {}).get("fee_type_id")
    conn   = get_db()
    fee    = conn.execute(
        "SELECT * FROM fee_types WHERE id=?", (fee_id,)
    ).fetchone()
    if not fee:
        conn.close()
        return jsonify(success=False, message="Invalid fee type")
    ref = txn_ref()
    conn.execute(
```

```
        "INSERT INTO payments "
        "(transaction_ref,student_id,fee_type_id,amount,status) "
        "VALUES (?,?,?,?,'pending')",
        (ref, session["student_id"], fee_id, fee["amount"])
    )
    conn.commit()
    conn.close()
    return jsonify(
        success=True, transaction_ref=ref,
        amount=fee["amount"], fee_name=fee["name"]
    )
@app.route("/api/receipt/<ref>")
@login_required
def download_receipt(ref):
    conn = get_db()
    pay = conn.execute(
        "SELECT * FROM payments "
        "WHERE transaction_ref=? AND student_id=? AND status='success'",
        (ref, session["student_id"])
    ).fetchone()
    conn.close()
    if not pay or not pay["receipt_filename"]:
        return "Receipt not found", 404
    return send_file(
        os.path.join(RECEIPTS_DIR, pay["receipt_filename"]),
        as_attachment=True,
        download_name=f"GOUNI_Receipt_{pay['receipt_number']}.pdf"
    )
```

```
# ── ADMIN API ──────────────────────────────────────────────────────────

@app.route("/api/admin/stats")
@admin_required
def admin_stats():
    conn = get_db()
    total_students  = conn.execute(
        "SELECT COUNT(*) FROM students"
    ).fetchone()[0]
    cleared         = conn.execute(
        "SELECT COUNT(*) FROM clearance_status "
        "WHERE is_financially_cleared=1"
    ).fetchone()[0]
    total_revenue   = conn.execute(
        "SELECT COALESCE(SUM(amount),0) FROM payments "
        "WHERE status='success'"
    ).fetchone()[0]
    pending         = conn.execute(
        "SELECT COUNT(*) FROM payments WHERE status='pending'"
    ).fetchone()[0]
    success_count   = conn.execute(
        "SELECT COUNT(*) FROM payments WHERE status='success'"
    ).fetchone()[0]
    webhook_count   = conn.execute(
        "SELECT COUNT(*) FROM webhook_logs"
    ).fetchone()[0]
    conn.close()
    return jsonify(
```

```
        totalStudents=total_students, clearedStudents=cleared,
        totalRevenue=total_revenue,   pendingPayments=pending,
        successPayments=success_count, webhookCount=webhook_count
    )
@app.route("/api/admin/payments")
@admin_required
def admin_payments():
    conn = get_db()
    rows = conn.execute("""
        SELECT p.*, s.full_name, s.reg_number, s.department,
               s.level, ft.name AS fee_name
        FROM payments p
        JOIN students s ON p.student_id = s.id
        JOIN fee_types ft ON p.fee_type_id = ft.id
        ORDER BY p.created_at DESC LIMIT 100
    """).fetchall()
    conn.close()
    return jsonify([dict(r) for r in rows])
@app.route("/api/admin/students")
@admin_required
def admin_students():
    conn = get_db()
    rows = conn.execute("""
        SELECT s.*, cs.is_financially_cleared, cs.cleared_at
        FROM students s
        LEFT JOIN clearance_status cs ON s.id = cs.student_id
        ORDER BY s.full_name
    """).fetchall()
```

```
    conn.close()
    return jsonify([dict(r) for r in rows])
@app.route("/api/admin/webhooks")
@admin_required
def admin_webhooks():
    conn = get_db()
    rows = conn.execute(
        "SELECT * FROM webhook_logs ORDER BY processed_at DESC LIMIT 50"
    ).fetchall()
    conn.close()
    return jsonify([dict(r) for r in rows])
@app.route("/api/admin/receipt/<ref>")
@admin_required
def admin_receipt(ref):
    conn = get_db()
    pay = conn.execute(
        "SELECT * FROM payments "
        "WHERE transaction_ref=? AND status='success'", (ref,)
    ).fetchone()
    conn.close()
    if not pay or not pay["receipt_filename"]:
        return "Receipt not found", 404
    return send_file(
        os.path.join(RECEIPTS_DIR, pay["receipt_filename"]),
        as_attachment=True,
        download_name=f"GOUNI_Receipt_{pay['receipt_number']}.pdf"
    )
```

APPENDIX B:

Database Seed Data

The system is pre-seeded with the following demonstration data:

STUDENTS

Reg. Number	Full Name	Department	Level
CS/2021/001	Chukwuemeka Obi	Computer Science	400
CS/2021/002	Adaeze Nwachukwu	Computer Science	400
CS/2022/015	Kelechi Eze	Computer Science	300
MTH/2021/008	Ngozi Okeke	Mathematics	400
CS/2023/042	Chinedu Igwe	Computer Science	200

DEMO FEE TYPES

FEE TYPE	AMOUNT (₦)
SCHOOL FEES	285,000.00
ACCEPTANCE FEE	50,000.00
HOSTEL FEE	120,000.00
EXAM DOCKET FEE	15,000.00
LATE REGISTRATION FEE	25,000.00

ADMIN ACCOUNTS

Username	Role
bursary_admin	Bursary Officer
ict_admin	ICT Director

(Default password for all demo accounts: students use "student123", admins use "admin123")