**Blockchain Banking System**

**ABSTRACT:**

The "Blockchain Banking System" project is an innovative financial management system built on the principles of blockchain technology. This system employs a decentralized and secure ledger to record banking transactions,providing users with a transparent and tamper-resistant environment for managing their accounts. The project incorporates key features such as user authentication, account creation, fund transfers, and transaction history, all orchestrated within the framework of a blockchain.In this project, each financialtransaction is encapsulated in a block, and a series of interconnected blocks form a secure and immutable chain. The use of blockchain not only ensures the integrity of the financial data but also introduces a novel approach to banking that reduces the risk of fraud and enhances transparency. The system allows users to create accounts, perform transactions, and view their transaction history seamlessly. Additionally, the blockchain implementation includes features for notifications and account deletion, contributing to the overall user experience.The "Blockchain Banking System" project serves as a demonstration of the potential for blockchain technology in revolutionizing traditional banking systems. Through its practical implementation, this project showcases the benefits of decentralized ledgers in enhancing security, transparency, and user control in the realm of financial management.

**CHAPTER-1**

**INTRODUCTION:**.

The "Blockchain Banking System" project is an innovative financial management system built on the principles of blockchain technology.

This system employs a decentralized and secure ledger to record banking transactions,providing users with a transparent and tamper-resistant environment for managing their accounts. The project incorporates key features such as user authentication, account creation, fund transfers, and transaction history, all orchestrated within the framework of a blockchain.In this

project, each financialtransaction is encapsulated in a block, and a series of interconnected blocks form a secure and immutable chain. The use of blockchain not only ensures the integrity of the financial data but also introduces a novel approach to banking that reducesthe risk of fraud and enhances transparency. The system allows users to create accounts, perform transactions, and view their transaction history seamlessly. Additionally, the blockchain implementation includes features for notifications and account deletion, contributing to the overall user experience.The "Blockchain Banking System" project serves as a demonstration of the potential for blockchain technology in revolutionizing traditional banking systems. Through its practical implementation, this project showcases the benefits of decentralized ledgers in enhancing security, transparency, and user control in the realm of financial management.

Top of Form

**EXISTING SYSTEM:**

The current banking system relies on centralized databases to manage user accounts and financial transactions. In this conventional approach, banks maintain control over a central server that stores sensitive user data and transaction records. However, this system is not without its challenges. Security concerns arise due to the vulnerability of a single point of failure, making it a target for potential cyber attacks. Furthermore, the lack of transparency and the reliance on manual processes contribute to inefficiencies, causing delays in transactions and hindering the overall user experience.The "Blockchain Banking System" project aims to address these inherent disadvantages by leveraging decentralized ledger technology. Through the implementation of a blockchain-based approach, the project seeks to enhance security, transparency, and efficiency in financial transactions, providing a more resilient and user-friendly alternative to the limitations of the existing centralized banking systems.

**DISADVANTAGES:**

* A breach in the central database could lead to unauthorized access, data manipulation, or even the compromise of sensitive customer information.
* The lack of transparency in traditional banking systems results in challenges for users to independently verify the accuracy of their transactions.
* This dependency on intermediaries not only contributes to delays but also increases the likelihood of errors and discrepancies in the transaction process.

**PROPOSED SYSTEM:**

The "Blockchain Banking System" is designed to revolutionizethetraditional banking landscape by integratingblockchain technology, offering a secure, transparent, and efficient financial ecosystem. In this proposed system, each user transaction is encapsulated within a block, andthe blocks are securely linked throughcryptographic hashes, creating an immutable ledger. This decentralized ledger ensures that all transactions are transparent, traceable, and resistant to tampering, providing an unprecedented level of security and trust for users.Furthermore, the proposed system incorporates smart contractsenabling self-executing agreements that automate various banking processes. Smart contracts not only enhance the speed of transaction settlements but also reduce the need for intermediaries, leading to lower transaction costs. The system empowers users with real-time visibility into their financial activities, fostering a user-centric approach to banking.

**ADVANTAGES:**

* The blockchain technology underlying the system ensures a high level of security by employing cryptographic techniques, reducing the risk of fraud and unauthorized access.
* The decentralized ledger provides transparent and traceable records of all transactions, instilling confidence in users by allowing them to verify their financial activities in real-time.
* Integration of smart contracts automates processes, reducing the need for intermediaries and minimizing transaction costs, ultimately providing a cost-effective banking solution.

**SOFTWARE SPECIFICATION:**

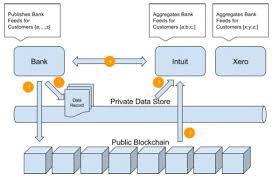
**HARDWARE CONFIGURATION:**

* Processor - i5
* Speed - 3 GHz
* RAM - 8 GB(min)S
* Hard Disk - 500 GB
* Key Board -Standard Windows Keyboard
* Mouse - Two or Three Button Mouse
* Monitor - SVGA

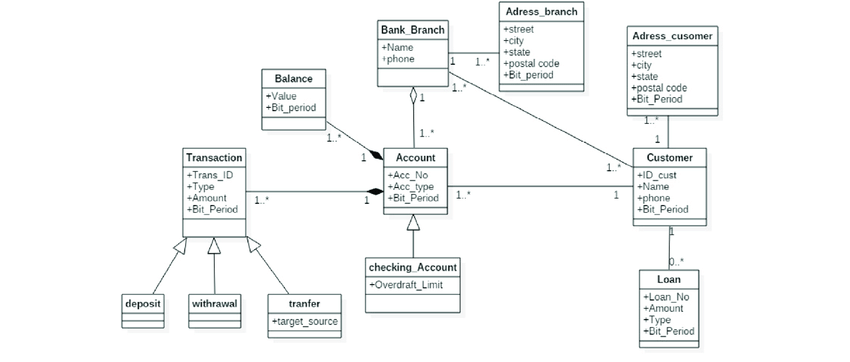
**SOFTWARE CONFIGURATION:**

* Operating System: Linux, Windows/7/10
* Tools: Anaconda, Jupiter, vscode
* Front End: HTML, CSS
* Server side Script: Python , AIML

**SYSTEM ARCHITECTURE DIAGRAM:**



**CLASS DIAGRAM:**

****

**DATAFLOW DIAGRAM:**

Employee login

Updateing balance

Money transation

User view balance

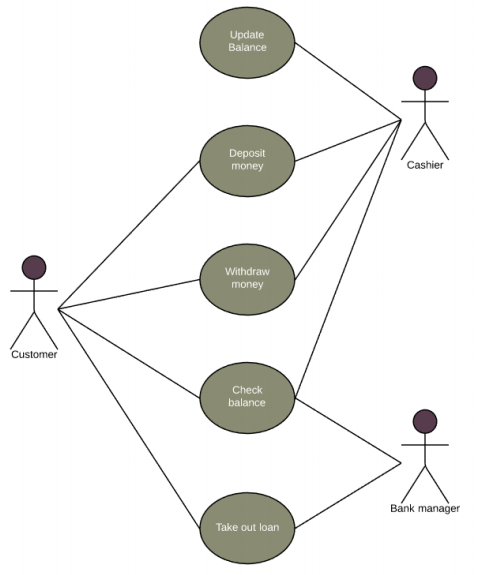
User transations

Employee accept

accpt

User login

**USE CASE DIAGRAM:**



empoyee

Admin

user

**ER DIAGRAM:**

Emplpyee login

Transation

User login

User data

User page

Logout

**CHAPTER-2**

**RELATED WORK:**

The project utilizes Django, a powerful web framework for Python, to implement user registration (reg), login (log), and logout (out) functionalities. User authentication is reinforced through the integration of Django's authentication system. Notably, the project distinguishes between regular users and employees, providing specific functionalities and views based on the user's role.The banking functionalities are well-implemented, allowing users to create accounts, check balances (balance), initiate transactions (transaction\_page), and view transaction histories (trans\_history). The system incorporates a thoughtful approach to account creation, utilizing notifications (create\_account\_noti) to streamline the account setup process.One of the notable features is the integration of blockchain technology to create a decentralized ledger for transactions. The Block class encapsulates key properties of a block, and functions such as calculate\_hash and create\_new\_block facilitate the creation of blocks within the blockchain. The use of the blockchain adds transparency and security to the transaction history, as each block is linked to the previous one, forming an immutable chain.It's commendable that the project handles edge cases, such as insufficient funds during transactions, and provides appropriate error messages..In summary, the "blockchain\_banking\_system" project showcases a well-implemented combination of Django and blockchain technology to create a secure and decentralized banking system with essential user functionalities. The integration of a blockchain ledger adds an innovative layer to traditional banking, fostering transparency and security in transaction processing.

**LITERATURE SURVEY REFERENCES:**

**Title:** " BLOCKCHAIN TECHNOLOGY –A BOON FOR THE BANKING SECTOR TO ENSURE SECURE TRANSACTION"

**Authors:** Assistant Professor, Department of Commerce

**Published in:** 2022

**Abstract:**

In the era of the digital economy, everyone is forced to convert to a digital transaction and payment modes. Though it has many merits to its list, there is a critical element called security which is a significant concern for both the service provider and the end-user. But the technological development gave solutions for the problem through the Blockchain method. Hence this chapter provides a holistic understanding of the application of blockchain technology in the banking sector. This chapter will discuss the meaning, steps, types, and application of blockchain technology in the banking sector. This also addresses banks' success stories in applying blockchain technology and challenges encountered during the implementation process. After reading this chapter, the learner will understand blockchain technology and its role in ensuring secured banking transactions.

**Title:** " Banking System Using Blockchain"

**Authors:** Ahlam Ansari , Hatim Shabbir Bohra, Daanish Kathi

**Published in:** 2022

**Abstract:** Blockchain is a distributed ledger technology which aids in recording transaction and trackingcapital in any business network. The main feature of blockchain is transparency. Our project is going to helpthese banks because we will be implementing blockchain ledger for creating a banking system in Linux.

Blockchain will boost trust, security, translucence, and the traceability of data shared across a bankingsystem and will be cost saving with new efficiencies. Because it provides rapid, shareable, and entirely clearinformation recorded on an immutable ledger that can only be viewed by authorised network users,

blockchain is the ideal option for distributing dataBanking systems are transferring from their older methods to more recentledger methods. Banking sector has started experimenting with blockchain ledger by copying current assettransactions on the blockchain

**Title:** Blockchain Revolution in Banking Industry"

**Authors:** Thulya Palihapitiya

**Published in:** 2020

**Abstract:** Abstract— today, banks are affected by economic and digital transformation, financial innovations and development of internet. Blockchain technology with cryptocurrency is underlying technology with promising application in the banking sector. Therefore, Aim of this paper is to do a research with the impact of Blockchain platform in the banking industry. To understand this technology, this research is to analyze technology functions with the model and anatomy of Blockchain architecture. Many researches for Blockchain technology are carried out consensus algorithms and four of them are discussed on this paper. How banking industry deal with this platform with advantages and limitations are mainly discussed in this paper

**Title:** " Blockchain Application in Banking System"

**Authors:** [Khairunnahar Suchana](https://www.scirp.org/journal/articles.aspx?searchcode=Khairunnahar++Suchana&searchfield=authors&page=1), [Syed Md Eftekhar Alam](https://www.scirp.org/journal/articles.aspx?searchcode=Syed+Md+Eftekhar++Alam&searchfield=authors&page=1), [Mohammad Monirujjaman Khan](https://www.scirp.org/journal/articles.aspx?searchcode=Mohammad+Monirujjaman++Khan&searchfield=authors&page=1)

**Published in:** 2021

**Abstract:**

Blockchain is one of those new and revolutionary technologies that will have a significant impact on the market and industry. In layman’s terms, Blockchain is a data structure that stores transactional records while also ensuring security, transparency, and decentralization. There is a digital signature on every transaction on a blockchain, which proves the authenticity of the blockchain. In a Blockchain, data is stored which is tamper-proof and cannot be changed as it uses encryption and digital signatures.. This paper will demonstrate transacting over a secure, blockchain-based network and therefore eliminate the need for intermediary entities. This paper is a review-based paper that provides the application and opportunities of the Blockchain in the banking system..

# Titile:” BLOCKCHAIN TECHNOLOGY IN BANKING SECTOR”

**Authors:** C.Mallesha\* S.Haripriya

**Published in:** 2019

**Abstract:**.

Blockchain technology is core, underlying technology with promising application prospects in the banking industry. With the increasing need for modernization in our day-to-day lives, people are open to accepting new technologies. From using a remote for controlling devices to using voice notes for giving commands, modern technology has made space in our regular lives. A study was conducted to identify the transparency of currency without third-party entering. It is the study about the blockchain technology framework and banking Industry. The major role is played in banking sector and main challenges are included. Blockchain technology is reshaping the future of Banking.

.

**CHAPTER– 3**

**MODULE:**

* **BankUsers Module**
* **Account Module**
* **Notification Module**
* **Block Module**
* **Transaction Module**

**MODULE DISCRIPTION:**

**BankUsers Module:**

The BankUsers model represents individual users in the banking system. It includes fields such as username, password, email, and phone to store essential information about each user. This model serves as the foundation for user authentication and account management within the system.

**Account Module:**

The Account model is linked to the BankUsers model through a foreign key relationship. It includes fields like account\_number and balance to manage individual bank accounts associated with users. This model tracks the account-specific details, such as the account number and current balance.

**Notification Module:**

The Notification model, also linked to BankUsers through a foreign key, represents notifications related to account creation. The status field indicates whether a user has been notified or not. This model is useful for managing the process of notifying users about the creation of their bank accounts.

**Block Module:**

The Block model is part of the blockchain implementation within the banking system. It includes fields like index, timestamp, data, hash, and previous\_hash. The index represents the position of the block in the blockchain, timestamp records when the block was created, data holds information about the block, and hash stores the block's unique identifier. The previous\_hash field links blocks in a chain, ensuring the integrity of the blockchain.

**Transaction Module:**

The Transaction model facilitates the recording of financial transactions between users. It includes fields like send\_id and receiver\_id, both referencing the BankUsers model through foreign keys. The time field captures the timestamp of the transaction, and trans\_amount stores the amount involved in the financial exchange. This model allows the system to maintain a comprehensive record of user transactions, enabling transparency and accountability in financial activities.

**CHAPTER 4**

**LANGUAGE DESCRIPTION:**

**PYTHON:**

Python is an interpreter, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding; make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.It ranges from simple automation tasks to gaming, web development, and even complex enterprise systems. These are the areas

where this technology is still the king with no or little competence: Machine learning as it has a plethora of libraries implementing machine learning algorithms.Python is a one-stop shop and relatively easy to learn, thus quite popular now. What other reasons exist for such universal popularity of this programming language and what companies have leveraged its opportunities to the max? Let’s talk about that. Python technology is quite popular among programmers, but the practice shows that business owners are also Python development believers and for good reason. Software developers love it for its straightforward syntax and reputation as one of the easiest programming languages to learn. Business owners or CTOs appreciate the fact that there’s a framework for pretty much anything – from web apps to machine learning. Moreover, it is not just a language but more a technology platform that has come together through a gigantic collaboration from

thousands of individual professional developers forming a huge and peculiar community of aficionados. So what is python used for and what are the tangible benefits the language brings to those who decided to use it? Below we’re going to discover that. Productivity and Speed It is a widespread theory within development circles that developing Python applications is approximately up to 10 times faster than developing the same application in Java or C/C++. The impressive benefit in terms of time saving can be explained by the clean object-oriented design, enhanced process control capabilities, and strong integration and text processing capacities. Moreover, its own unit testing framework contributes substantially to its speed and productivity.

**PYCHARM:**

PyCharm is a dedicated Python Integrated Development Environment (IDE) providing a wide range of essential tools for Python developers, tightly integrated to create a convenient environment for productive Python, web, and data science development.

Choose the best PyCharm for you﻿

**PyCharm is available in three editions:**

* Community (free and open-sourced): for smart and intelligent Python development, including code assistance, refactorings, visual debugging, and version control integration.
* Professional (paid) : for professional Python, web, and data science development, including code assistance, refactorings, visual debugging, version control integration, remote configurations, deployment, support for popular web frameworks, such as Django and Flask, database support, scientific tools (including Jupyter notebook support), big data tools.
* Edu (free and open-sourced): for learning programming languages and related technologies with integrated educational tools.

|  |  |  |
| --- | --- | --- |
| Requireme | Minimum | Recommended |

|  |  |  |
| --- | --- | --- |
| RAM | 4 GB of free RAM | 8 GB of total system RAM |
| CPU | Any modern CPU | Multi-core CPU. PyCharm supports multithreading for different operations and processes making it faster the more CPU cores it can use. |
| Disk space | 2.5 GB and another 1 GB for caches | SSD drive with at least 5 GB of free space |
| Monitor resolution | 1024x768 | 1920×1080 |
| Operating system | Officially released 64-bit versions of the following:   * Microsoft Windows 8 or later * macOS 10.13 or later * Any Linux distribution that supports Gnome, KDE, or Unity DE. PyCharm is not available for some Linux distributions, such as RHEL6 or CentOS6, that do not include [GLIBC](https://ftp.gnu.org/gnu/libc/) 2.14 or later.   Pre-release versions are not supported. | Latest 64-bit version of Windows, macOS, or Linux (for example, Debian, Ubuntu, or RHEL) |

**Supported languages﻿:**

To start developing in Python with PyCharm you need to download and install Python from python.org depending on your platform.

**SUPPORTED PLATFORMS﻿:**

[**Jupyter Notebook**](https://doc.cocalc.com/jupyter.html#id13)**:**

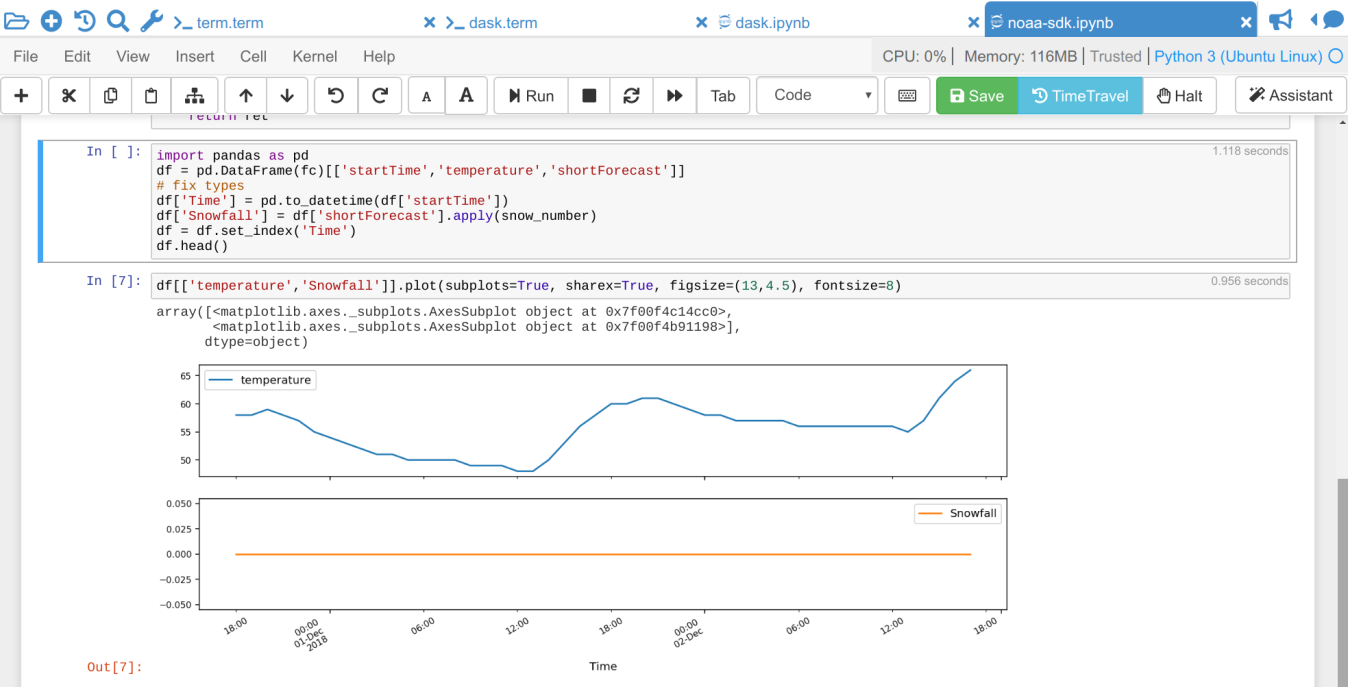
A Jupyter notebook is a specific filetype with the ending .ipynb, which records an interactive session with a **Kernel**. It made up of cells, which can either store one or more lines of code or formatted text. When you run a cell – which evaluates the piece of code in the cell via the active kernel session – you can see its output after the calculation is done. This combination of communicating back and forth with a kernel and adding descriptive text makes this form of document very attractive.

## [Jupyter Kernels](https://doc.cocalc.com/jupyter.html#id14):

You can choose the programming language and environment by selecting a Jupyter kernel for the notebook. Popular choices are [Python3](https://docs.python.org/3/), [SageMath](https://www.sagemath.org/), and [R](https://www.r-project.org/about.html). There many others. Our page on [Jupyter Kernel Selection](https://doc.cocalc.com/howto/jupyter-kernel-selection.html) shows how to set the kernel.

**JUPYTER NOTEBOOK BASIC:**

By default, a Jupyter notebook on CoCalc has all CoCalc’s core features, including real-time collaboration, side chat, and TimeTravel. Read more in our [blogpost](http://blog.sagemath.com/jupyter/2017/05/05/jupyter-rewrite-for-smc.html). The basic user interface looks like the following:

[](https://doc.cocalc.com/_images/jupyter-notebook-cocalc-1.png)

* Above the main area is a menu bar and a button row:
* The menu bar contains all commands, and in particular the Kernel menu is for changing it if necessary.
* The button row gives you a one-click access to Run the current cell (otherwise press your Shift+Return keys), a way to restart the kernel (which clears the current session) and a Save button to make sure CoCalc has stored the file. The Time Travel button allows you to see previous versions of that notebook, such that you can go back in time to recover from a bad change
* Active cell: in the screenshot above, the blue bar on the left and a blue border around a cell indicates that this is the currently active one. Actions like Run, Delete Cell, etc. operate on the currently selected cell. It is also possible to select more than one cell.
* Execution counter: On the left of each cell, there is an execution counter The number increases each time a cell is being run. After the kernel stopped and restarted, that counter starts again at 1.
* The output of code cells is below the input cell. For example, is the output of cell In the right hand corner of the input cell is some information about how long it took to calculate the result.
* Text cells are slightly different. Select “Markdown” in the dropdown menu in the button bar to change a code cell to such a markdown text cell. There, you can use Markdown to format the text. Similar to code-cells, either Run these text cells to see the processed Markdown code or press Shift+Return. To edit a text cell, either double click it or press your Return key.
* Saving: more general, the nice things about Jupyter Notebooks is that they save all your intput and output in one single file. This means you can download or publish the notebook as it is, and everyone else sees it in exactly the same way

**ANACONDA PYTHON:**

Anaconda® is a package manager, an environment manager, a Python/R data science distribution, and a collection of [over 7,500+ open-source packages](https://docs.anaconda.com/anaconda/packages/pkg-docs/). Anaconda is free and easy to install, and it offers [free community support](https://groups.google.com/a/anaconda.com/forum/?fromgroups#!forum/anaconda).

Get the Anaconda Cheat Sheet and then [download Anaconda](https://www.anaconda.com/downloads).

Want to install conda and use conda to install just the packages you need? Get [Miniconda](http://conda.pydata.org/miniconda.html).

**Anaconda Navigator or conda?**

After you install Anaconda or Miniconda, if you prefer a desktop graphical user interface (GUI) then use [Navigator](https://docs.anaconda.com/anaconda/navigator/). If you prefer to use Anaconda prompt (or terminal on Linux or macOS), then use that and conda. You can also switch between them.

You can install, remove, or update any Anaconda package with a few clicks in Navigator, or with a single conda command in Anaconda Prompt (terminal on Linux or macOS).

* **To try Navigator**, after installing Anaconda, click the Navigator icon on your operating system’s program menu, or in Anaconda prompt (or terminal on Linux or macOS), run the command anaconda navicator
* **To try conda**, after installing Anaconda or Miniconda, take the [20-minute conda test drive](https://conda.io/projects/conda/en/latest/user-guide/getting-started.html) and download a [conda cheat sheet](https://docs.conda.io/projects/conda/en/latest/user-guide/cheatsheet.html)

**Packages available in Anaconda:**

* Over 7,500 additional open-source packages (including R) can be individually installed from the Anaconda repository with the conda install command.
* Over [250 packages](https://docs.anaconda.com/anaconda/packages/pkg-docs/) are automatically installed with Anaconda
* Thousands of other packages are available from [Anaconda.org](https://anaconda.org/).
* You can download other packages using the pip install command that is installed with Anaconda. [Pip packages](https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-pkgs.html#installing-non-conda-packages) provide many of the features of conda packages and in some cases they can work together. However, the preference should be to install the conda package if it is available.
* You can also make your own [custom packages](https://conda.io/projects/conda-build/en/latest/) using the conda build command, and you can share them with others by uploading them to [Anaconda.org](http://anaconda.org/), PyPI, or other repositories.

**Previous versions:**

* Previous versions of Anaconda are available in the [archive](https://repo.anaconda.com/archive/). For a list of packages included in each previous version, see [Old package lists](https://docs.anaconda.com/anaconda/packages/oldpkglists/).
* Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, it does not matter which one you download, because you can create new environments that include any version of Python packaged with conda. See [Managing Python with conda](https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-python.html).

**CHAPTER-5**

**SYSTEM TESTING AND MAINTENANCE:**

Testing is vital to the success of the system. System testing makes a logical assumption that if all parts of the system are correct, the goal will be successfully achieved. In the testing process we test the actual system in an organization and gather errors from the new system operates in full efficiency as stated. System testing is the stage of implementation, which is aimed to ensuring that the system works accurately and efficiently.

In the testing process we test the actual system in an organization and gather errors from the new system and take initiatives to correct the same. All the front-end and back-end connectivity are tested to be sure that the new system operates in full efficiency as stated. System testing is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiently.The main objective of testing is to uncover errors from the system. For the uncovering process we have to give proper input data to the system. So we should have more conscious to give input data. It is important to give correct inputs to efficient testing.

Testing is done for each module. After testing all the modules, the modules are integrated and testing of the final system is done with the test data, specially designed to show that the system will operate successfully in all its aspects conditions. Thus the system testing is a confirmation that all is correct and an opportunity to show the user that the system works. Inadequate testing or non-testing leads to errors that may appear few months later.

This will create two problems, Time delay between the cause and appearance of the problem. The effect of the system errors on files and records within the system. The purpose of the system testing is to consider all the likely variations to which it will be suggested and push the system to its limits.The testing process focuses on logical intervals of the software ensuring that all the statements have been tested and on the function intervals (i.e.,) conducting tests to uncover errors and ensure that defined inputs will produce actual results that agree with the required results. Testing has to be done using the two common steps Unit testing and Integration testing. In the project system testing is made as follows:

The procedure level testing is made first. By giving improper inputs, the errors occurred are noted and eliminated. This is the final step in system life cycle. Here we implement the tested error-free system into real-life environment and make necessary changes, which runs in an online fashion. Here system maintenance is done every months or year based on company policies, and is checked for errors like runtime errors, long run errors and other maintenances like table verification and reports.Integration Testing is a level of software testing where individual units are combined and tested as a group.The purpose of this level is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration testing.

**METHOD:**

Any of Black Box Testing, White Box Testing, and Gray Box Testing methods can be used. Normally, the method depends on your definition of ‘unit’.

**TASKS:**

* Integration Test Plan
* Prepare
* Review
* Rework
* Baseline
* Integration Test Cases/Scripts
* Prepare
* Review
* Rework
* Baseline
* Integration Test
* Perform
* **Overview**

The provided Django-based Blockchain Banking System incorporates user authentication, account management, and blockchain functionality for secure financial transactions. For this project, a comprehensive software testing strategy is crucial to ensure the reliability and security of the system.

**Software Testing**

**Unit Testing:**

Unit tests should be written to evaluate the functionality of critical functions and components in the code. Key areas for unit testing include:

**Blockchain Logic:**

Verify the correctness of the calculate\_hash function.

Test the creation of the genesis block using create\_genesis\_block.

Validate the creation of new blocks with create\_new\_block.

**User Authentication:**

Test user registration (reg view) to ensure new users are created correctly.

Validate user login (log view) with correct and incorrect credentials.

Confirm proper handling of user logout (out view).

**Account Management:**

Test the creation of account notifications (create\_account\_noti view).

Verify the creation of user accounts based on notifications (account\_create view).

Confirm the deletion of user accounts (delete view).

**Financial Transactions:**

Test financial transactions (transaction\_page view) to ensure proper deduction and addition of balances.

Validate the creation of transaction records (trans model).

Confirm the creation of new blocks in the blockchain for each transaction.

**Integration Testing:**

Integration tests should be performed to assess the interaction between different components and modules of the system. Key integration test scenarios include:

**User Journey:**

Test the complete user journey from registration to transaction history retrieval.

Validate the redirections between different views.

**Account Management:**

Ensure the integration between the notification, account, and bankusers models is functioning correctly.

Validate the relationship between notifications and user accounts.

**Blockchain Integration**

Confirm the integration of blockchain logic with financial transactions.

**End-to-End Testing**

End-to-end tests should be conducted to simulate real-world scenarios and assess the system's functionality as a whole. Key end-to-end test cases include:

**User Registration and Login**

Test user registration and login with various scenarios, including invalid credentials.

**Financial Transactions**

Simulate financial transactions with different amounts and validate the resulting balances.

**Account Creation and Deletion**

Test the creation of user accounts based on notifications and verify account deletion.

**Blockchain Functionality**

Validate the creation of new blocks for each financial transaction.

**CHAPTER – 6**

**METHODOLOGY:**

1. **Introduction:**

The Blockchain Banking System is designed to provide secure and transparent financial transactions through the use of blockchain technology. This methodology outlines the key steps and approaches used in the development of this system.

**2. Software Development Life Cycle (SDLC)**

**2.1 Requirement Analysis:**

Identified the need for a secure and transparent banking system.

Defined user stories, including user authentication, account management, and blockchain-based transactions.

**2.2 Design:**

Created models (bankusers, account, notification, Block, transaction) to represent users, accounts, notifications, blocks, and transactions.

Implemented blockchain logic for secure and transparent financial transactions.

Designed views and templates for user registration, login, account management, and transaction functionalities.

**2.3 Implementation:**

Developed the system using Django, a high-level Python web framework.

Implemented user authentication using Django's built-in authentication system.

Utilized Django models for database management.

Integrated blockchain logic for creating and managing blocks.

**2.4 Testing**

Conducted unit, integration, and end-to-end testing to ensure the correctness and reliability of the system.

Tested various scenarios, including user registration, login, financial transactions, and blockchain functionality.

Documented test cases and executed tests to identify and fix bugs.

**2.5 Deployment**

Deployed the system on a web server to make it accessible to users.

Configured settings for security, performance, and scalability.

Monitored the deployed system for any issues and performed necessary updates.

**3. Blockchain Implementation**

**3.1 Genesis Block:**

Created the genesis block during system initialization with index 0, containing basic information.

**3.2 Block Creation**

Implemented a function to calculate the hash for each block based on its index, previous hash, timestamp, and data.

Created new blocks for each financial transaction, linking them to the previous block for a secure and chronological chain.

### ****Technology Overview:****

The Blockchain Banking System utilizes a combination of web development technologies and blockchain principles to create a secure and transparent banking experience. Below is an overview of the technologies and frameworks employed in the development of this system:

**1. Django Framework**

Django is a high-level Python web framework that follows the Model-View-Controller (MVC) architectural pattern. It provides a clean and pragmatic design for developing web applications quickly. Key features include:

Models: Used for defining database models such as bankusers, account, notification, Block, and transaction.

Views: Handle user interface logic, rendering HTML templates, and processing user input.

Templates: HTML templates are used to render dynamic content and generate user interfaces.

Authentication: Django's built-in authentication system is employed for user registration, login, and account management.

Database Management: Django provides an Object-Relational Mapping (ORM) system to interact with the database seamlessly.

**2. Blockchain Implementation**

The system integrates blockchain principles for secure and transparent financial transactions. The following components represent the basic blockchain architecture:

**2.1 Blocks Class**

The Blocks class represents a block in the blockchain and contains attributes such as index, previous\_hash, timestamp, data, and hash.

**2.2 Hash Calculation**

The calculate\_hash function uses the SHA-256 hashing algorithm to generate a hash based on the block's index, previous\_hash, timestamp, and data.

**2.3 Genesis Block**

The create\_genesis\_block function initializes the blockchain with a genesis block containing basic information.

**2.4 Block Creation**

The create\_new\_block function creates a new block by calculating its hash and linking it to the previous block.

**3. Random Module**

The random module is utilized to generate random account numbers during the account creation process.

**4. Timezone and Datetime**

The timezone module is used to handle timestamp information, and the datetime module is employed to record transaction times.

**5. Django QuerySet and Q Object**

The filter method with the Q object is employed to perform complex queries on Django models, allowing for efficient retrieval of data.

**6. Technologies for Future Consideration**

While the current system focuses on a Django-based implementation with basic blockchain principles, future enhancements may involve integrating more advanced blockchain technologies such as Ethereum for a decentralized and distributed ledger.

This technology overview highlights the key components and frameworks that contribute to the functionality of the Blockchain Banking System. The combination of Django and blockchain principles ensures a robust and secure banking experience for users.

**CHAPTER – 7**

**CONCLUSION:**

The Blockchain Banking System achieves a harmonious integration of Django's web development capabilities and blockchain technology to create a secure and transparent banking experience. Users can register, manage accounts, perform transactions, and view transaction history seamlessly. The inclusion of blockchain principles enhances the security of financial transactions by introducing a decentralized and tamper-resistant ledger.Future enhancements to this system may involve the integration of more advanced blockchain technologies, such as smart contracts and decentralized applications (DApps), to further explore the potential of decentralized finance (DeFi). Additionally, ongoing testing, debugging, and user feedback will contribute to refining and optimizing the system for real-world use.In conclusion, the Blockchain Banking System stands as a testament to the innovative possibilities that arise from the convergence of web development and blockchain technologies, offering a glimpse into the future of secure and transparent financial systems.

**CHAPTER - 8**

**REFERENCE:**

**1**,BLOCKCHAIN TECHNOLOGY – A BOON FOR THE BANKING SECTOR TO ENSURE SECURE TRANSACTIONAuthor: [Author Name], Assistant Professor, Department of CommercePublished in: 2022

**2**.Banking System Using Blockchain,Authors: Ahlam Ansari, Hatim Shabbir Bohra, Daanish KathiPublished in: 2022

**3**.Blockchain Revolution in Banking IndustryAuthor: Thulya PalihapitiyaPublished in: 2020

**4.**Blockchain Application in Banking SystemAuthors: Khairunnahar Suchana, Syed Md Eftekhar Alam, Mohammad Monirujjaman KhanPublished in: 2021

**5**.BLOCKCHAIN TECHNOLOGY IN BANKING SECTORAuthors: C. Mallesha, S. Haripriya Published in: 2019

**CODEING:**

**Views.py**

from django.shortcuts import render, redirect

from .models import bankusers, account, notification, Block, transaction

from django.contrib.auth import authenticate,login,logout

import random

from django.utils import timezone

import hashlib

from datetime import datetime

from django.db.models import Q

class Blocks:

def \_\_init\_\_(self, index, previous\_hash, timestamp, data, hash):

self.index = index

self.previous\_hash = previous\_hash

self.timestamp = timestamp

self.data = data

self.hash = hash

def calculate\_hash(index, previous\_hash, timestamp, data):

value = str(index) + str(previous\_hash) + str(timestamp) + str(data)

return hashlib.sha256(value.encode()).hexdigest()

def create\_genesis\_block():

genesis\_block = Block.objects.create(index=0, timestamp=timezone.now(), data="Genesis Block", hash="0", previous\_hash="0")

return genesis\_block

def create\_new\_block(previous\_block, data, request):

index = previous\_block.index + 1

timestamp = timezone.now()

hash\_value = calculate\_hash(index, previous\_block.hash, timestamp, data)

new\_block = Block.objects.create(user=request.user, index=index, timestamp=timestamp, data=data, hash=hash\_value, previous\_hash=previous\_block.hash)

return new\_block

def home(request):

return render(request, 'index.html')

def reg(request):

if request.method == 'POST':

name = request.POST.get('name')

password = request.POST.get('password')

email = request.POST.get('email')

number = request.POST.get('number')

user = bankusers.objects.create(username= name, password= password, email= email, phone= number)

user.set\_password(password)

user.save()

login(request, user)

return render(request, 'profile.html')

return render(request, 'Register.html')

def log(request):

if request.method == 'POST':

name = request.POST.get('name')

password = request.POST.get('password')

user = authenticate(request, username = name, password = password)

if user is not None:

login(request, user)

if user.role == 'employee':

data = notification.objects.filter(status = None)

print(data)

return render(request, 'emp.html', {'data': data})

return render(request, 'profile.html', {'user': user})

return render(request, 'login.html')

def out(request):

logout(request)

return render(request, 'index.html')

def main(request):

user = request.user

return render(request, 'profile.html', {'user': user})

def balance(request):

check\_account = account.objects.filter(user\_id=request.user.id).first() # Use first() to get a single instance or None

print(check\_account)

if check\_account:

return render(request, 'balance.html', {'account': check\_account})

else:

return render(request, 'profile.html', {'message': 'First Create Account'})

def create\_account\_noti(request):

check\_account=account.objects.filter(user\_id = request.user.id)

check\_noti=notification.objects.filter(user = request.user.id)

print(check\_account, check\_noti)

if check\_account.exists() or check\_noti.exists():

return render(request, 'profile.html', {'message': 'You Already Have A Account'})

else:

notify = notification.objects.create(user = request.user)

notify.save()

return render(request, 'profile.html')

def account\_create(request, flag):

id = request.GET.get('id')

save = notification.objects.get(id = id)

save.status = flag

check\_account=account.objects.filter(user\_id = request.user.id)

if check\_account.exists():

data = notification.objects.filter(status = None)

return render(request, 'emp.html', {'data': data})

else:

account\_number = random.randint(10\*\*11, 10\*\*12 - 1)

account\_create = account.objects.create(user\_id = save.user, account\_number = account\_number)

account\_create.save()

save.save()

data = notification.objects.filter(status = None)

return render(request, 'emp.html', {'data': data})

def transaction\_page(request):

if request.method == 'POST':

account\_number = request.POST.get('account')

amount = float(request.POST.get('amount'))

logged\_in\_user = account.objects.get(user\_id = request.user)

if logged\_in\_user.balance < amount:

return render(request, 'transaction.html', {'error\_message': 'Insufficient funds'})

try:

target\_account = account.objects.get(account\_number=account\_number)

except bankusers.DoesNotExist:

return render(request, 'transaction.html', {'error\_message': 'Invalid account number'})

logged\_in\_user.balance -= amount

target\_account.balance += amount

trans = transaction.objects.create(send\_id = request.user, receiver\_id = target\_account.user\_id, time = datetime.now().time(), trans\_amount = amount)

if Block.objects.exists():

# Get the latest block if it exists

latest\_block = Block.objects.latest('index')

else:

# Create the genesis block if no blocks exist

latest\_block = create\_genesis\_block()

data = f"Transaction: {amount} from {logged\_in\_user.user\_id} to {target\_account.user\_id}"

new\_block = create\_new\_block(latest\_block, data, request)

# Save the new block to the database

new\_block.save()

logged\_in\_user.save()

target\_account.save()

trans.save()

return redirect('main')

return render(request, 'transaction.html')

def trans\_history(request):

data = transaction.objects.filter(Q(send\_id=request.user) | Q(receiver\_id=request.user))

return render(request, 'Transactionh.html', {'data': data})

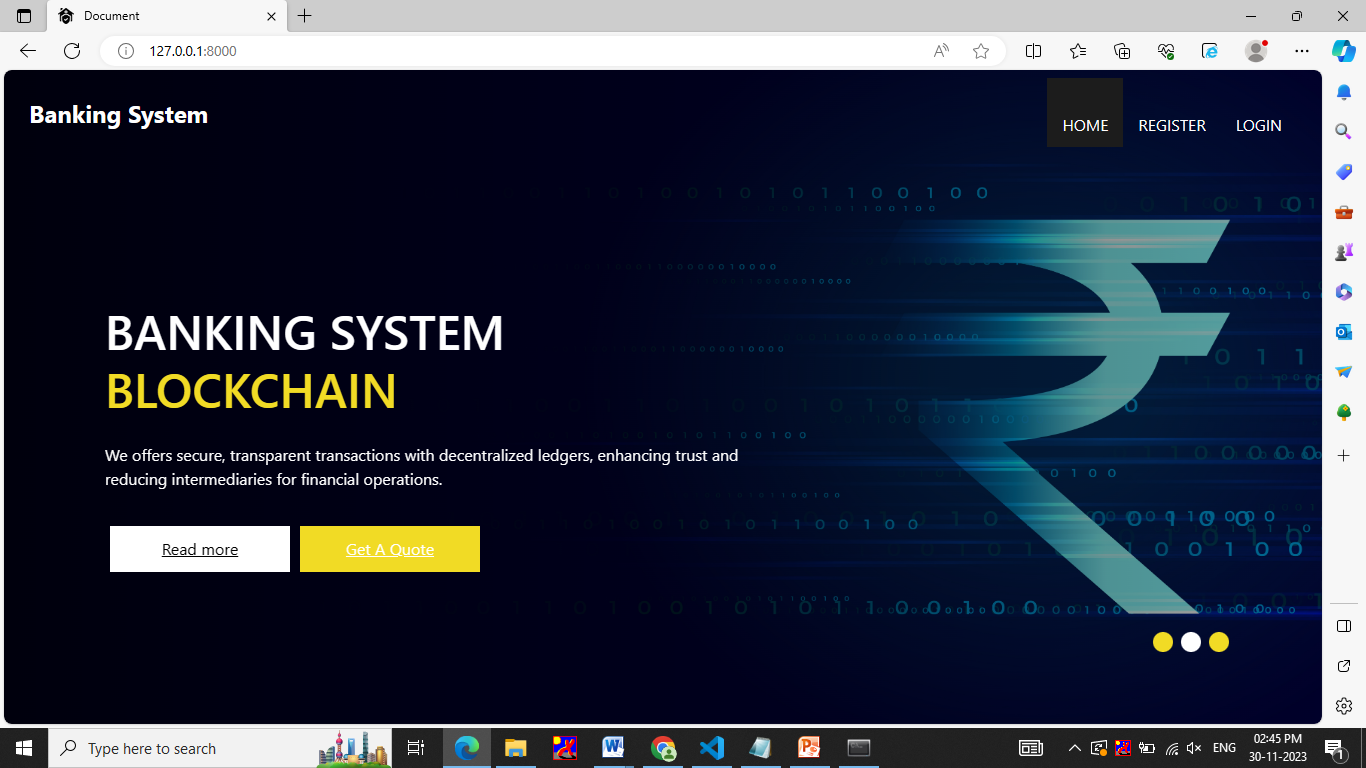
def delete(request):

del\_account = account.objects.delete(user\_id = request.user)

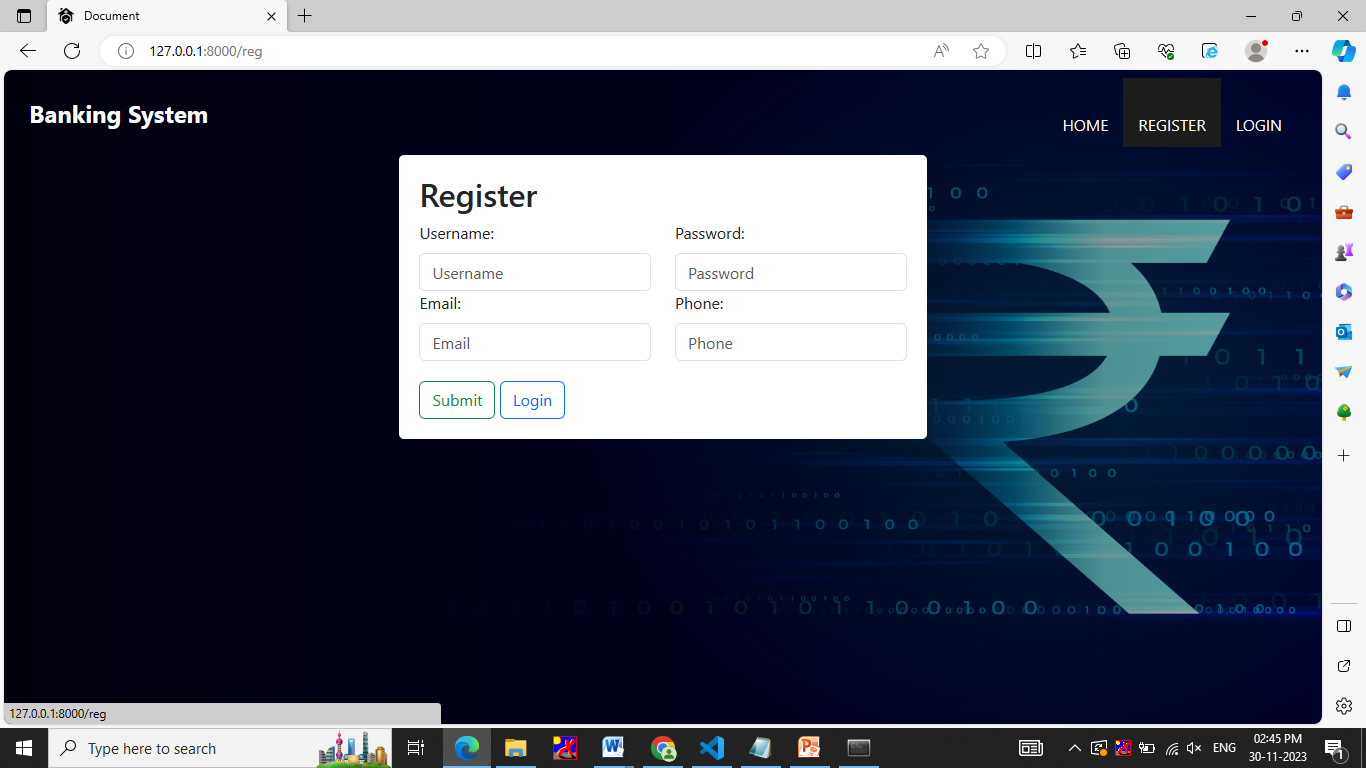
return render(request, 'profile.html')

**SCREENSHOTS:**

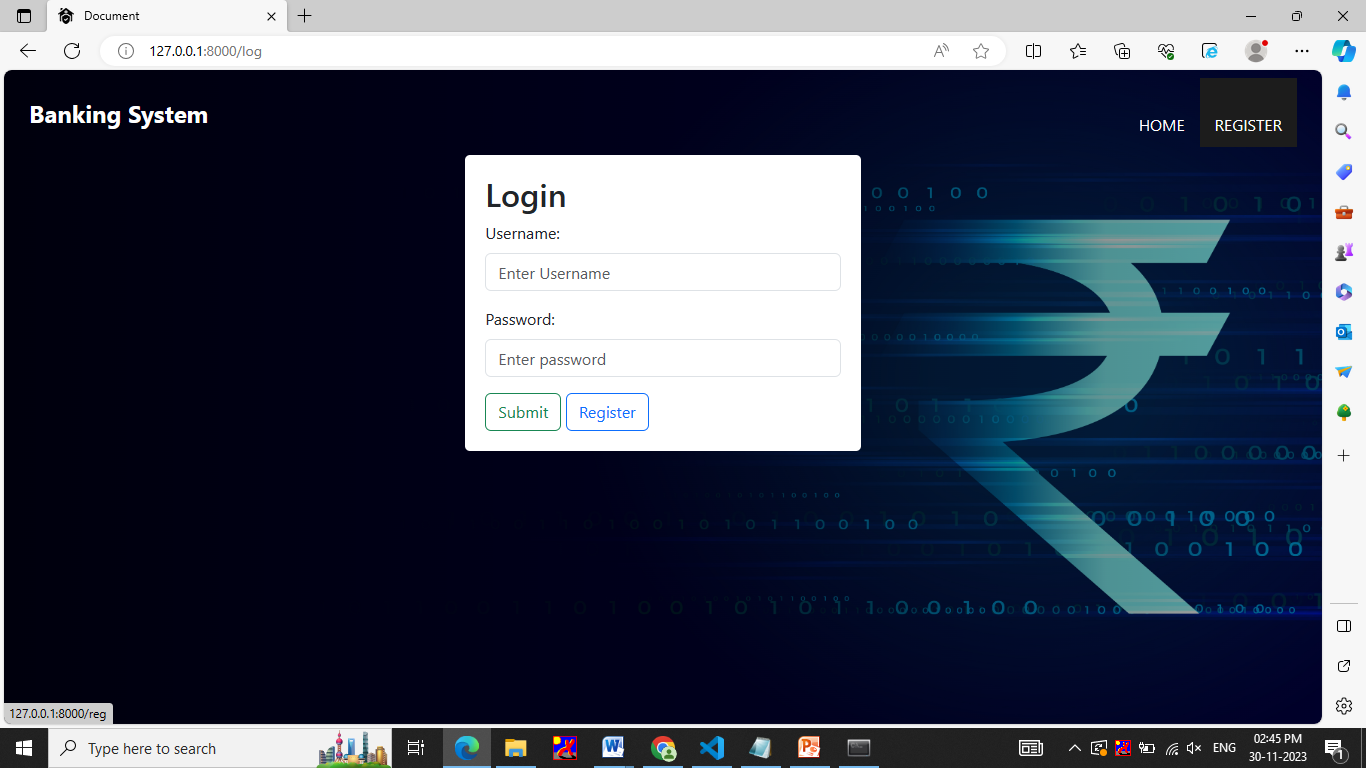
HOME PAGE



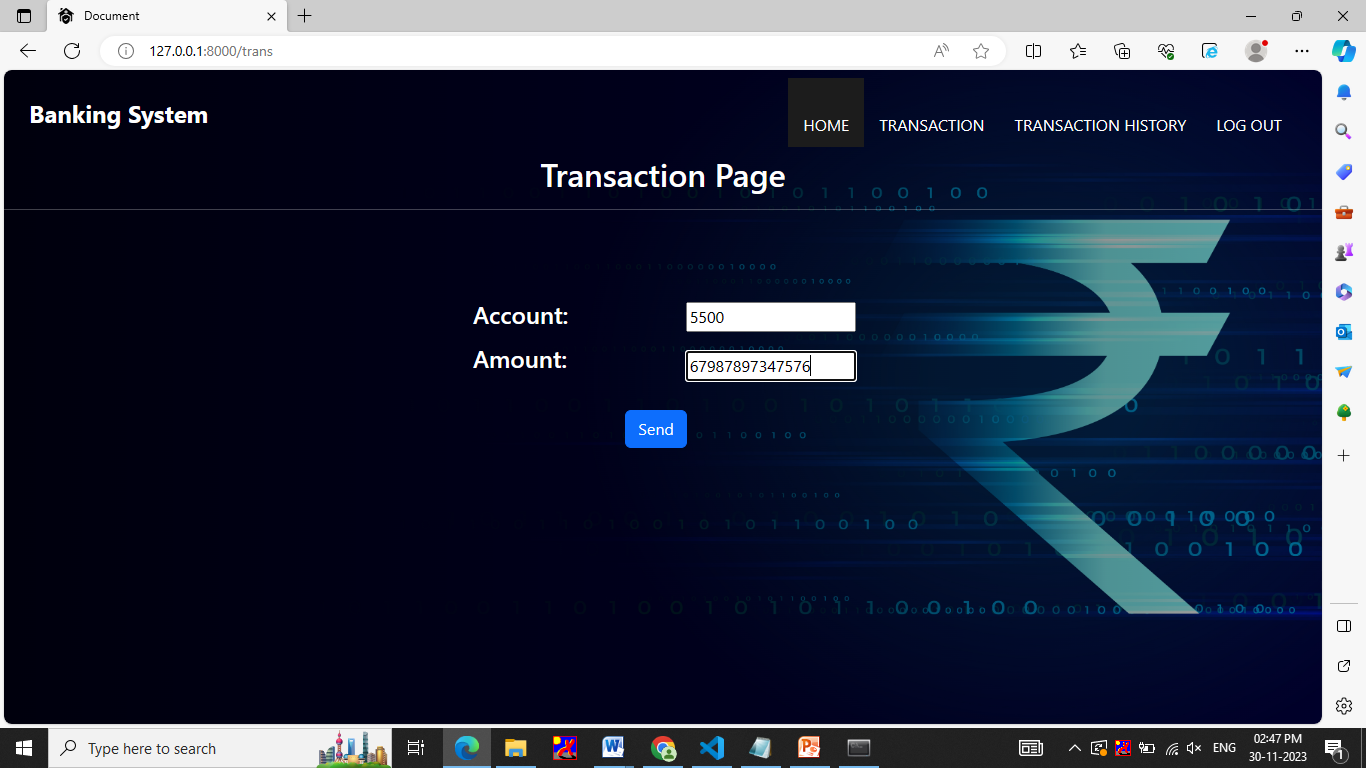
Register page



Login page



Transaction page



Employee page

