

ANUDIP FOUNDATION

Project title

“Banking System”

By

Name	Enrollment No.
Pradnya Raju Birdawade	AF0481781

Under Guidance

Of

Rajshri Thete

Mam

ABSTRACT

The **Banking System** project is a comprehensive web-based application designed to provide a range of banking services to customers and administrative functionalities to bank administrators. Developed using **Python**, the **Django Framework**, and **MySQL Database**, the system aims to ensure security, efficiency, and ease of use for financial operations such as deposits, withdrawals, money transfers, and account management. This system is intended to provide an accessible, secure, and scalable solution for modern banking needs, leveraging the powerful capabilities of the Django framework for web development, MySQL for database management, and Python for backend logic.

Overview and Objective

The primary objective of this project is to develop an automated banking platform that can handle core banking operations while providing a user-friendly interface for both customers and administrators. Customers can access their accounts to carry out common banking tasks, such as checking balances, depositing funds, withdrawing money, and transferring money between accounts. Administrators, on the other hand, have additional privileges to manage users, monitor transactions, and ensure the smooth operation of the system. The project focuses on creating a **secure, robust, and reliable** application that ensures data integrity, prevents unauthorized access, and performs real-time processing of financial transactions.

By utilizing a **web-based** approach, the system ensures that users can access their accounts and perform transactions from anywhere at any time, provided they have an internet connection. This approach makes banking more accessible and convenient, aligning with the growing trend of digital banking.

ACKNOWLEDGEMENT

The project “**Banking System**” is the Project work carried out by

Name	Enrollment No
Pradnya Raju Birdawade	AF0481781

Under the Guidance.

We are thankful to my project guide for guiding me to complete the Project.

His suggestions and valuable information regarding the formation of the Project Report have provided me a lot of help in completing the Project and its related topics.

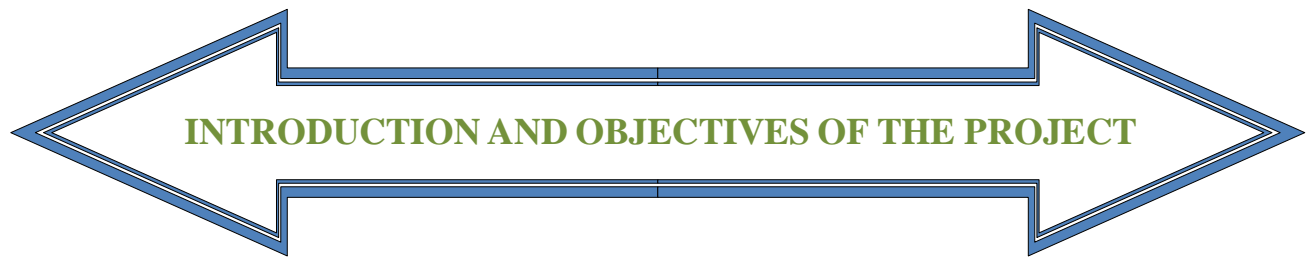
We are also thankful to my family member and friends who were always there to provide support and moral boost up.



SR.NO	Contents
1	Introduction
2	Objective
3	System Analysis
4	Feasibility Study
5	Project Planning & Scheduling
6	Software Requirement Specification
7	System Design
8	Module Descriptions
9	Coding & Implementation Summary
10	Testing and Security Measures
11	Cost Estimation
12	Future Enhancements
13	Conclusion
14	Bibliography



Banking System



INTRODUCTION

The Banking System is a secure, web-based application developed using Python, Django, and MySQL that facilitates the essential functionalities of online banking for customers and administrative staff. With the evolution of digital technology and the growing demand for real-time financial services, this system aims to digitize core banking operations such as user account management, fund transfers, transaction tracking, and administrative oversight.

In today's fast-paced environment, traditional banking processes often face challenges like long queues, delayed processing, manual errors, and restricted service hours. The proposed Banking System addresses these inefficiencies by providing users with seamless access to banking services from any internet-enabled device, 24/7. It simplifies complex banking procedures, reduces dependency on physical infrastructure, and increases user satisfaction by offering transparency and instant operations.

This system enables customers to securely register, log in, and manage their banking operations such as deposits, withdrawals, balance inquiries, and inter-account fund transfers. Each transaction is logged in the system with a timestamp to maintain accuracy and transparency. The application ensures role-based access control, where administrative users can manage user profiles, monitor transaction logs, and handle

exception cases like unauthorized access or unusual activity.

Built on Django's robust MVC (Model-View-Controller, here represented as MVT – Model-View-Template) framework, the system is structured, modular, and highly maintainable. Data is stored and managed using MySQL, a relational database system known for its speed, scalability, and integrity. Django's built-in security mechanisms—such as CSRF protection, hashed password storage, and authentication middleware—ensure that user data is protected against common web vulnerabilities like SQL injection and cross-site scripting (XSS).

The user interface, developed using HTML, CSS, and Django templates, offers a clean and intuitive experience, enhancing accessibility even for non-technical users. Administrative operations are streamlined through Django's powerful admin dashboard, allowing efficient data handling and system monitoring.

Overall, this Banking System not only increases operational efficiency and user convenience but also sets a foundational platform for integrating more advanced features in the future, such as mobile banking, two-factor authentication, and credit analytics. It reflects the practical application of full-stack web development using open-source technologies to create scalable and secure financial solutions.

OBJECTIVES

The purpose of hotel booking system is to automate the existing manual system by the help of computerized equipment's and full-fledged computer software, fulfilling their requirement, so that their valuable or information can be stored for a longer period with easy accessing and manipulating of the same. The required software and hardware are easily available and easy to work with. This proposes that efficiency of hotel organizations could be improved by integrating service-oriented operations service-oriented operations with project management principles.

The primary objectives of the Hotel Management System project are:

1. **To automate hotel operations:** Replace manual processes with a digital system for managing bookings, check-ins, check-outs, and payments.
2. **To improve efficiency:** Minimize the time and effort required for staff to perform daily hotel operations through a centralized management interface.
3. **To enhance customer experience:** Allow guests to view available rooms, make online reservations, and receive booking confirmations conveniently.
4. **To maintain accurate records:** Store and manage guest data, room information, and transaction history in a secure and structured manner using a relational database.
5. **To generate reports:** Provide features for generating booking reports, occupancy summaries, and revenue insights to assist in business decision-making.



➤ **PROBLEM DEFINITION:**

Traditional banking systems are fraught with numerous limitations:

- **Manual Errors:** Human involvement in data entry and transaction processing leads to inconsistencies and inaccuracies in financial records.
- **Customer Inconvenience:** Long queues in physical bank branches discourage customers from accessing banking services promptly.
- **Limited Service Hours:** Conventional banking operations are confined to specific hours, leaving users without access during weekends, holidays, or after hours.
- **Lack of Transparency:** Users often lack real-time visibility into their account activities, leading to mistrust and confusion regarding transaction records.

The proposed system aims to address these issues by offering a digital alternative that integrates efficiency, transparency, and accessibility.

2. Objectives of the System

- Provide secure and user-friendly online access to banking features.
- Ensure real-time processing of transactions and updates.
- Improve the accuracy of transaction records through automated logging.
- Enhance customer trust through a transparent and detailed transaction history.
- Eliminate the need for physical bank visits by enabling comprehensive remote banking functionalities.

3. System Environment and Technology Stack

The system is built using:

- **Django Framework:** A powerful Python-based web framework that supports rapid development and clean design. It manages business logic, handles user authentication, and enforces secure access to features.
- **MySQL Database:** Acts as the backend storage for user accounts, transaction records, and system logs. It ensures data integrity and supports concurrent access through ACID-compliant transactions.

These technologies enable robust system performance, scalability, and maintainability.

4. Functional Requirements

The system provides the following functionalities:

- **User Authentication:** Secure login system for customers and administrators.
- **Account Management:** View and manage personal account details, change passwords, and update contact information.
- **Fund Transfers:** Transfer funds between accounts with instant verification of balances and recipients.
- **Balance Inquiry:** View real-time account balances.
- **Transaction Tracking:** Detailed view of transaction history with timestamps and status reports.
- **Admin Controls:** Manage users, monitor logs, and perform administrative tasks like resetting accounts or suspending services.

5. Non-Functional Requirements

- **Security:** Encrypted connections (e.g., HTTPS), secure password hashing, and input validation to prevent SQL injection or cross-site scripting.
- **Scalability:** Designed to support growing numbers of users and concurrent operations.

- Availability: 24/7 access enabled through a web interface, ensuring customers can access banking services anytime.
- Reliability: Stable system operations with minimal downtime and error handling mechanisms in place.

6. Benefits of the Proposed System

The proposed banking system significantly enhances the banking experience by automating and digitizing key services. Users benefit from faster services, accurate records, and complete control over their financial operations. Banks benefit from reduced workloads, fewer operational errors, and improved customer engagement.

❖ FEASIBILITY STUDY:

The banking system project was evaluated based on four key areas of feasibility: technical, economic, operational, and schedule. These aspects ensure the viability and sustainability of the system from development through deployment.

1. Technical Feasibility

This project was developed using the Python Django framework for the backend and MySQL as the database management system. Both are open-source technologies, meaning they are free to use and supported by active developer communities. Django is a high-level Python web framework that promotes rapid development and clean, pragmatic design. It comes with built-in security features (like protection against SQL injection, cross-site scripting, etc.) and a powerful ORM (Object-Relational Mapping) for database management, which makes development more efficient. MySQL, being a mature and reliable database system, ensures that the data storage layer is stable and scalable. These technologies are highly compatible and have been widely used in enterprise-grade applications, ensuring that the system can be maintained, scaled, and upgraded in the future.

2. Economic Feasibility

The development cost of this system was minimal due to the use of open-source tools such as Python, Django, and MySQL, which do not require licensing fees. Additionally, the system can be hosted on budget-friendly web hosting services or cloud platforms such as Heroku, PythonAnywhere, or even basic VPS servers. Since no proprietary software was used, the total investment primarily involved developer time, making this solution financially viable for startups, academic purposes, or small businesses. Future enhancements and scaling can be done gradually as user demand increases, keeping costs under control.

3. Operational Feasibility

The system is designed with a user-friendly interface for both end-users and administrators. The front-end ensures that customers can easily perform banking operations such as viewing account balances, making transactions, and managing their profiles. The admin panel allows for efficient management of users, transactions, and security controls. Furthermore, secure user authentication and role-based access controls

ensure that the system complies with essential data privacy and protection standards. This makes it practical for real-world use and suitable for training environments or small-scale deployments.

Steps in Conducting a Feasibility Study:

The primary objective of conducting a feasibility study for the **Banking System** is to evaluate whether the proposed system is practical and whether its development and implementation are economically viable. The feasibility study begins by analyzing the overall situation of the hotel and its existing operational processes. For example, it will assess the current challenges faced by hotel management, such as inefficient room booking systems, manual guest check-in processes, and difficulties in generating reports. The study will evaluate how the proposed system can address these challenges, streamline operations, and improve efficiency.

The second part of the feasibility study will focus on estimating the costs and benefits of the proposed **Banking System**. This will include analyzing the development costs, potential savings from automating processes, and the expected improvement in customer satisfaction. The study will also examine whether there are simpler or more cost-effective alternatives to the proposed system, such as improving the existing manual processes rather than implementing a complex software solution. If the proposed system is deemed both feasible and beneficial, the feasibility study will help in setting up a strategic implementation plan, including setting measurable goals and concrete steps for the system's development.

Typically, a feasibility study for this project should be conducted by a qualified consultant with expertise in the hospitality industry and software development. The consultant would gather the necessary information and data to ensure that the feasibility study is accurate and objective. It is also crucial for the hotel's internal team to be involved in the process, providing access to the necessary data about current operations, staff roles, and business requirements.

Advantages of making Feasibility study:

- **Comprehensive System Analysis:** The feasibility study helps in analyzing the complete requirements of the system, ensuring that all aspects, such as room booking, guest management, billing, and report generation, are thoroughly evaluated.
- **Risk Identification:** The study helps identify potential risks involved in the development, deployment, and operation of the system, allowing the hotel to plan for risk management and mitigation strategies.
- **Cost/Benefit Analysis:** The feasibility study provides a clear analysis of the costs and benefits, helping the hotel determine whether the investment in the system will result in sufficient returns through operational efficiency, reduced errors, and improved customer satisfaction.
- **Training and Implementation Plans:** The study highlights the need for training developers and staff members, ensuring that they are well-prepared for the system's implementation.

Different Types of Feasibility:

1. **Technical Feasibility:** This examines whether the necessary technology is available and suitable for the project. The study will analyze the technical requirements for building and running the hotel management system, including server configuration, database management systems, and the compatibility of technologies used in

Django, MySQL/PostgreSQL, and front-end tools (HTML, CSS, JavaScript).

2. **Schedule Feasibility:** This evaluates whether the project can be developed within the proposed timeline. The feasibility study will outline a realistic schedule for the design, development, testing, and deployment phases of the Hotel Management System, ensuring that the system can be implemented within the required time frame.
3. **Social Feasibility:** This considers whether the proposed system will be accepted by the hotel's staff and customers. It will involve gauging how employees and customers will react to the new system and whether they will find it easy to use and beneficial. This can include user training for staff and ensuring the user interface is intuitive for customers making bookings.
4. **Legal Feasibility:** This assesses whether the system complies with relevant legal requirements, such as data protection laws (e.g., GDPR) and financial regulations. The system must ensure secure data handling and proper consent collection from users.
5. **Marketing Feasibility:** This examines the market forces that might affect the adoption and success of the system. The study will look at the competitive landscape in the hospitality industry, understanding how other hotels manage operations and identifying how this system can provide a competitive advantage.

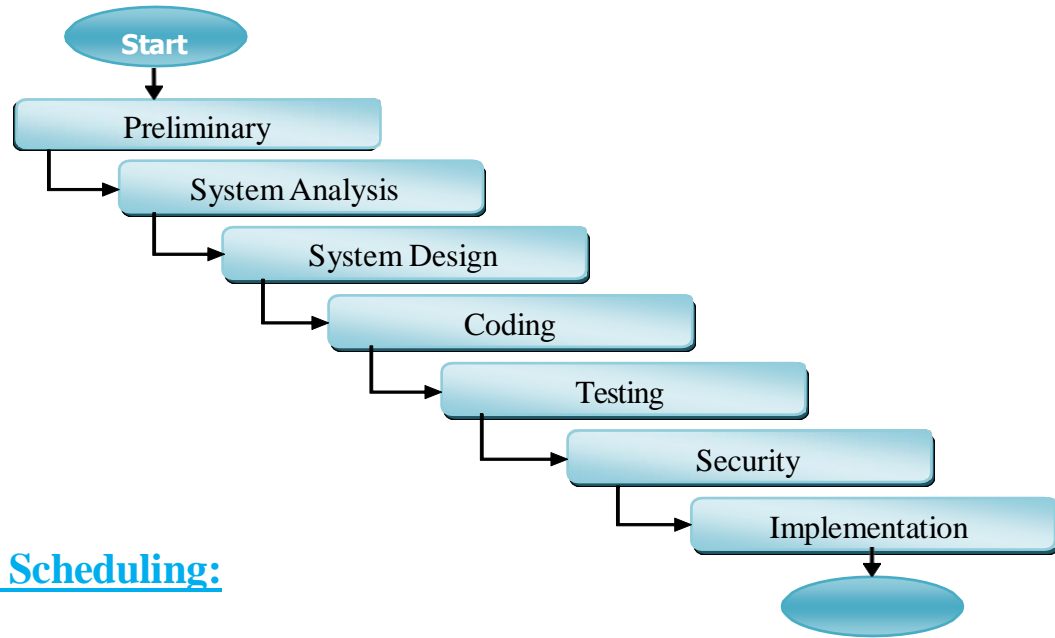
➤ PROJECT PLANNING AND SCHEDULING:

PROJECT PLANNING

■ Project Planning

■ Phase to Cover

1. Preliminary Investigation
2. System Analysis
3. System Design
4. Coding
5. Security
6. Testing
7. Implementation



Project Scheduling:

■ **Compartmentalization**

Activity Diagram:

Hotel Management System	
Preliminary Investigation	Identification of Need
	Requirement Specifications
	Feasibility Study
System Analysis	Project Planning & Scheduling
	SRS Preparations
	Software Engineering Paradigm
System Design	Module Description
	Database Design
	Procedural Design
	User Interface Design
Coding	Error free Coding
	Debugging
Testing	Unit Testing
	Integration Testing
	System Testing
Security	Application Security
	Database Security
Implementation	Implementation
	Documentation

➤ SOFTWARE REQUIREMENT SPECIFICATION (SRS):

1. Functional Requirements

These define the core functionalities that the system must deliver to meet user and business needs:

- **User Registration/Login**
Users must be able to create accounts by providing necessary credentials and log in securely. Authentication will be handled using Django's built-in system with support for password hashing and session management.
- **Account Dashboard**
After logging in, users should be redirected to a personalized dashboard displaying their account balance, recent transactions, and quick access to services like transfers and deposits.
- **Deposit and Withdraw Money**
Users can increase or decrease their account balance by initiating deposit and withdrawal operations, which will be reflected in real-time and logged in the transaction history.
- **Fund Transfer Between Accounts**
The system supports internal money transfers between registered users. Input validation, balance checks, and transaction recording are essential components of this feature.
- **Transaction History View**
Each user should have access to a paginated view of past transactions, filtered by date or type (deposit, withdrawal, transfer). This helps in financial tracking and transparency.
- **Admin Panel for User and Transaction Management**
An administrative backend enables the admin to monitor user accounts, approve or suspend access, and oversee all transaction logs for audit and regulatory purposes.

2. Non-Functional Requirements

These define the quality attributes of the system that enhance usability, security, and performance:

- **Usability**
The system should have a clean and responsive user interface designed using a frontend framework like Bootstrap. Navigation should be intuitive, with appropriate tooltips and error handling to assist users.

- **Reliability**

Django's ORM (Object-Relational Mapping) layer ensures that all database transactions are atomic and consistent, minimizing data corruption risks. The use of robust exception handling and rollback mechanisms further supports data integrity.

- **Security**

- All passwords will be stored using hashing algorithms like PBKDF2 or bcrypt.
- The application will use Django's built-in CSRF protection to prevent cross-site request forgery.
- Access control will be enforced via role-based authentication (admin vs. user).
- HTTPS will be used for secure data transmission during deployment.

- **Performance**

Query optimization will be done using Django's querysets and indexing in MySQL. For high-traffic environments, caching mechanisms (e.g., Memcached or Redis) can be introduced to reduce database load and speed up response times.

3. Hardware Requirements

- **Server-Side:**

- Minimum: Dual Core CPU @ 2.0 GHz, 2 GB RAM, 20 GB HDD
- Recommended: Quad Core CPU, 4 GB RAM, SSD storage for better performance
- Network: Stable internet connection, preferably with static IP for deployment

- **Client-Side:**

- Any modern web browser (Chrome, Firefox, Edge, Safari)
- Device with at least 1 GB RAM
- Internet connection (wired or wireless)

4. Software Requirements

- **Programming Language: Python 3.x**

Python is a versatile, readable language with extensive libraries, making it suitable for backend logic and integrations.

- **Framework: Django 3.x/4.x**

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It offers built-in features for admin panels, authentication, and database management.

- **Database: MySQL**

MySQL is a powerful, open-source relational database that integrates seamlessly with Django and offers scalability and reliability for financial data.

- **Web Server: Apache or Nginx (for deployment)**

Either Apache or Nginx can be used as a reverse proxy or standalone web server. These servers handle request routing, static file serving, and SSL configuration.

➤ SOFTWARE ENGINEERING PARADIGM:

The conventional and mostly used software engineering paradigm till now the Waterfall Model is applied on this project. But according to project complexity and nature slight deviation from the proposed original waterfall model is taken into consideration. The three characteristics of Waterfall Model as linear, rigid, and monolithic are diverted here.

The reasons and assumptions are explained below:

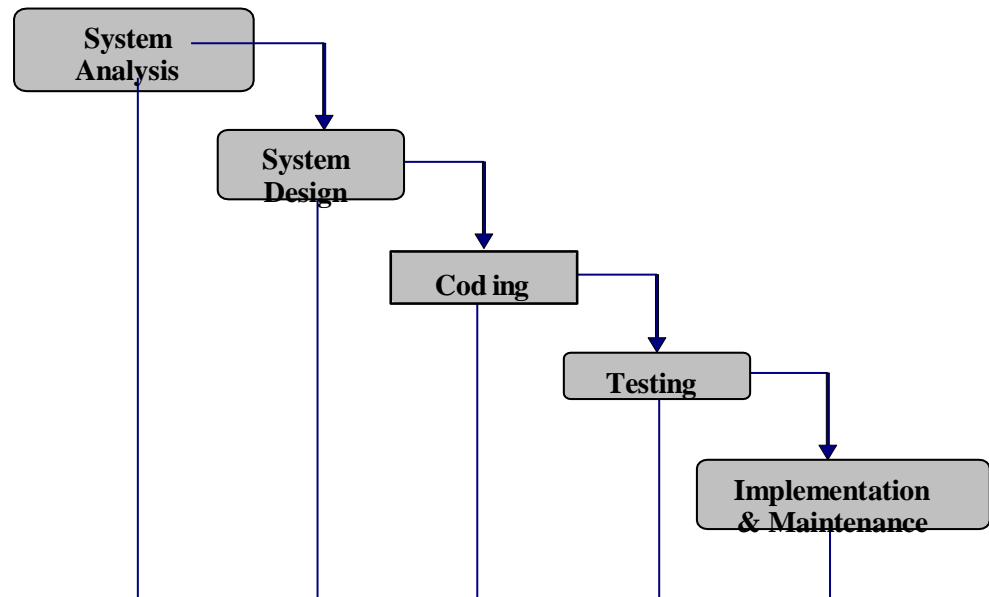
The linearity of Waterfall Model is based on the assumption that software development may proceed linearly from analysis down to coding. But, in this project feedback loops are applied to the immediately preceding stages.

Another underlying assumption of the Waterfall Model is phase rigidity – i.e., that the results of each phase are frozen before proceeding to the next phase. But, this characteristic is not strictly maintained throughout the project. Therefore overlapping of two or more phases is allowed according to needs and judgments.

Finally, the Waterfall Model is monolithic in the sense that all planning is oriented to single delivery date. But since this project has deadline imposed by the organization, planning is done into successive phases. This project is the term-end project and obviously need to submit as soon

as possible calendar date. With such considerations is selected the Waterfall model as the project development paradigm.

Water-fall Model with feedback mechanism for the proposed application as Software Engineering paradigm:



ANALYSIS DIAGRAMS

DFD (Data Flow Diagram):

In the late 1970s *data-flow diagrams (DFDs)* were introduced and popularized for structured analysis and design (Game and Sarsen 1979). DFDs show the flow of data from external entities into the system, showed how the data moved from one process to another, as well as its logical storage.

A **data flow diagram (DFD)** is a significant modeling technique for analyzing and constructing information processes. DFD literally means an illustration that explains the course or movement of information in a process. DFD illustrates this flow of information in a process based on the inputs and outputs. A DFD can be referred to as a Process Model.

Data flow diagrams can be used to provide a clear representation of any business function. The technique starts with an overall picture of the business and continues by analyzing each of the functional areas of interest.

As the name suggests, Data Flow Diagram (DFD) is an illustration that explicates the passage of information in a process. A DFD can be easily drawn using simple symbols. Additionally, complicated processes can be easily automated by creating DFDs using easy-to-use, free downloadable diagramming tools. A DFD is a model for constructing and analyzing information processes. DFD illustrates the flow of information in a process depending upon the inputs and outputs. A DFD can also be referred to as a Process Model. A DFD demonstrates business or technical process with the support of the outside data saved, plus the data flowing from the process to another and the end results.

Additionally, a **DFD** can be utilized to visualize data processing or a structured design. A DFD illustrates technical or business processes with the help of the external data stored, the data flowing from a process to another, and the results.

A designer usually draws a context-level DFD showing the relationship between the entities inside and outside of a system as one single step. This basic DFD can be then disintegrated to a lower level diagram demonstrating smaller steps exhibiting details of the system that is being

modeled. Numerous levels may be required to explain a complicated system.

Process

The process shape represents a task that handles data within the application. The task may process the data or perform an action based on the data.

Multiple Processes

The multiple process shape is used to present a collection of sub processes. The multiple processes can be broken down into its sub processes in another DFD.

External Entity

The external entity shape is used to represent any entity outside the application that interacts with the application via an entry point.

Data Flow

The data flow shape represents data movement within the application. The direction of the data movement is represented by the arrow.

Data Store

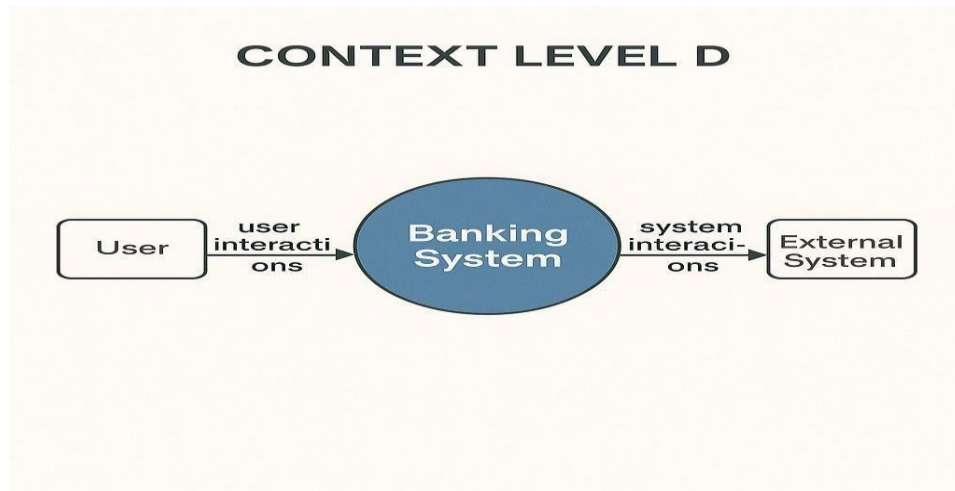
The data store shape is used to represent locations where data is stored. Data stores do not modify the data, they only store data.

Privilege Boundary

The privilege boundary shape is used to represent the change of privilege levels as the data flows through the application.

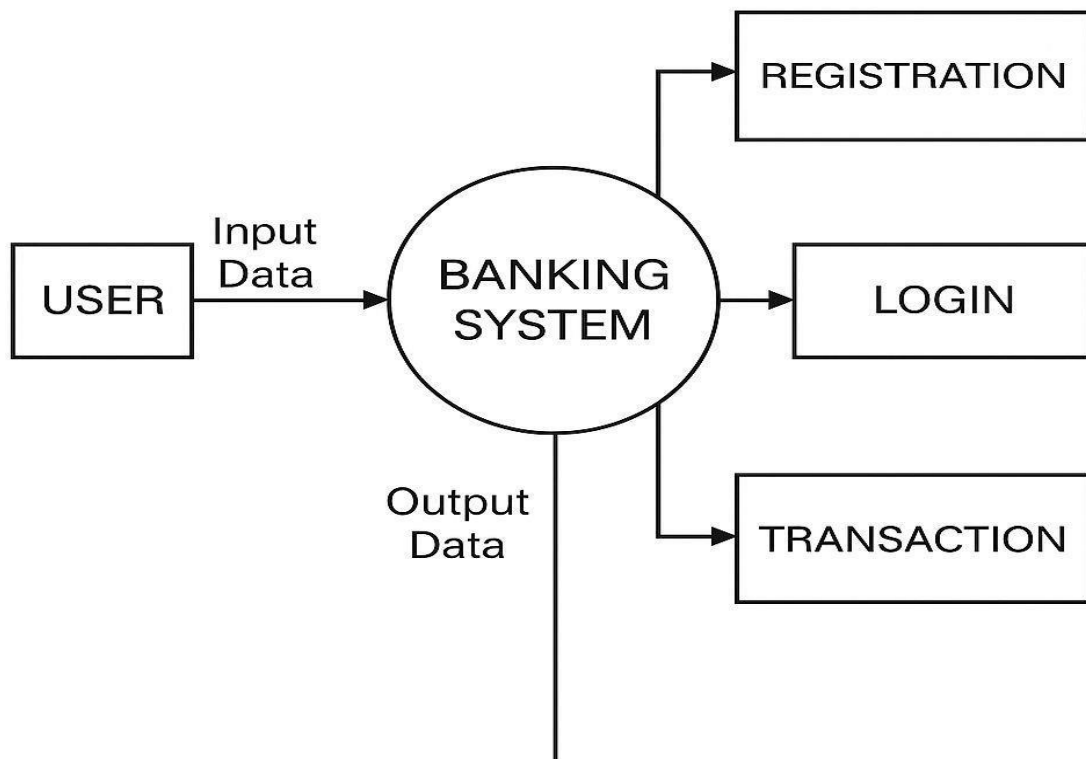
Examples of Data Flow Diagrams - These examples demonstrate how to draw data flow diagram.

Before it was eventually replaced, a copy machine suffered frequent paper jams and became a notorious troublemaker. Often, a problem could be cleared by simply opening and closing the access panel. Someone observed the situation and flowcharted the troubleshooting procedure used by most people.



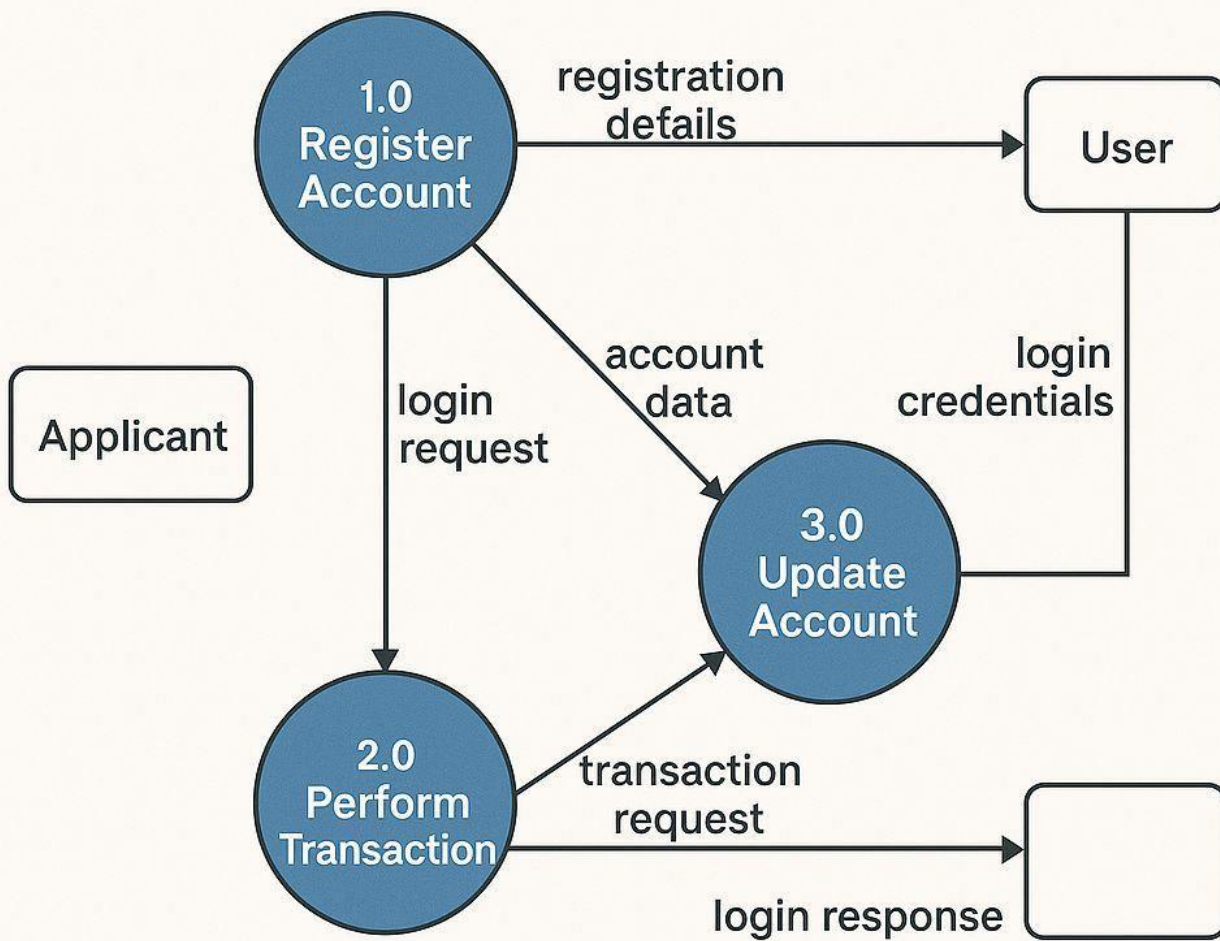
Level1 DFD Diagram:

BANKING SYSTEM

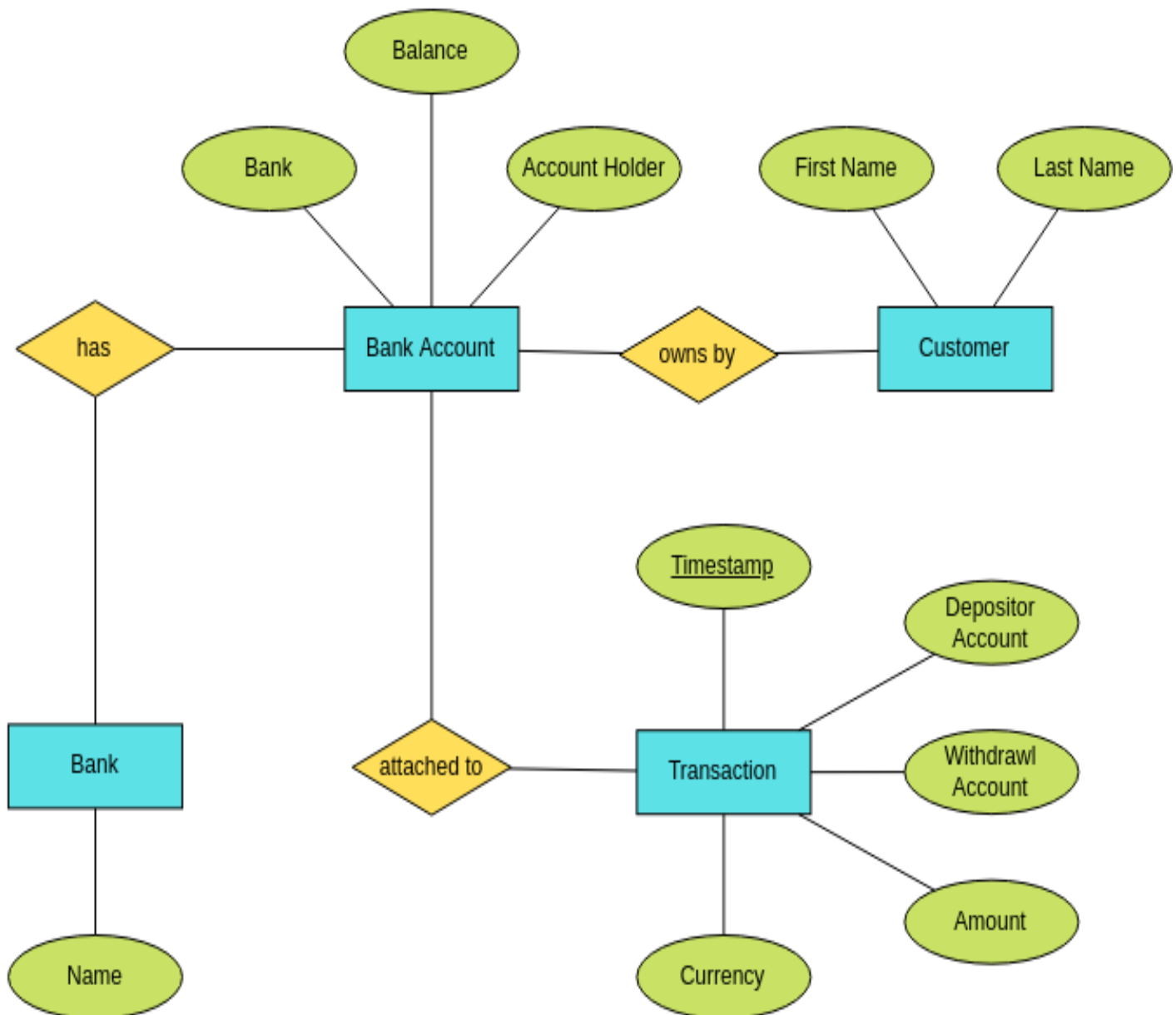


Level 2 DFD Diagram:

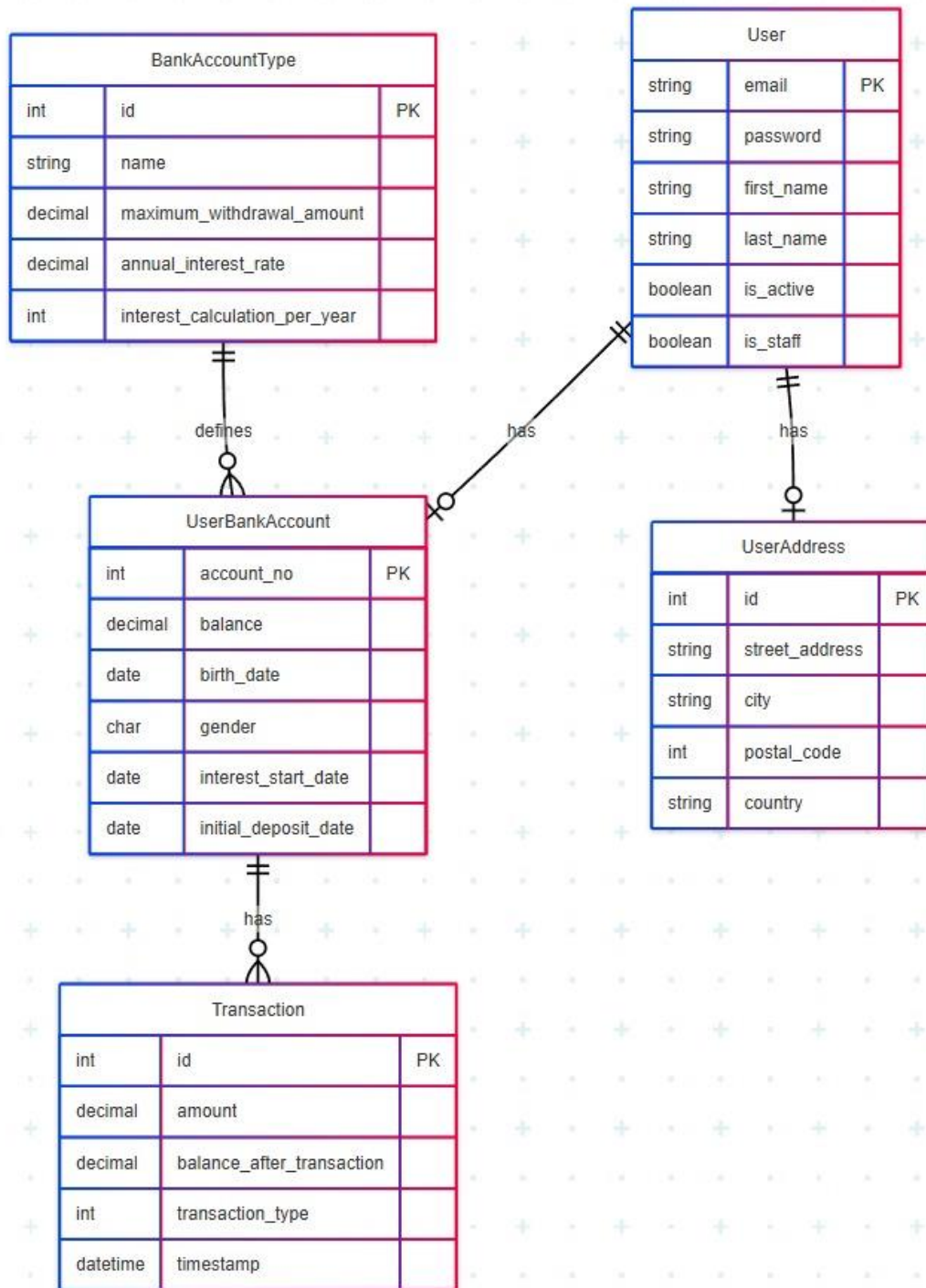
LEVEL 2 DFD



Entity Relationship Diagram (ERD):



Class Diagram:








The Banking System is structured using Django's MVT (Model-View-Template) architecture, which is a powerful and scalable design pattern used to separate concerns and maintain modular, maintainable code. Each layer of the MVT model plays a specific role in the request/response lifecycle.

1. Models (Data Layer)

The Models represent the core data structure of the application. Django uses its built-in Object-Relational Mapping (ORM) system to define models as Python classes, which are translated into database tables.

Key models include:

-  User
Extends Django's built-in User model to include fields like email, username, and optionally full name or role (admin/user). This model handles user authentication and login/logout functionality.
-  Account
Represents a user's bank account. Each account is linked to a user and includes fields such as account number, account type (e.g., savings, checking), current balance, and creation date.
-  Transaction
Tracks all monetary operations such as deposits, withdrawals, and transfers. Each transaction includes details such as sender account, receiver account (if applicable), amount, timestamp, and transaction type.

Model relationships (such as ForeignKey) are used to maintain database normalization and referential integrity. For example, each Account links to a User, and each Transaction links to one or more Accounts.

2. Views (Logic Layer)

Views in Django are Python functions or classes that process incoming HTTP requests, interact with the database via models, apply business logic, and return HTTP responses.

Examples of views:

- User registration and login views
- Dashboard view that fetches account and transaction data
- Deposit/withdrawal views that validate inputs and update account balances
- Fund transfer view that checks for sufficient balance and creates corresponding transaction entries

Views are often connected to templates and are protected by access control mechanisms using decorators like `@login_required` or permission checks for admin actions.

3. Templates (Presentation Layer)

Templates are HTML files with embedded Django Template Language (DTL) that render dynamic content to users. They display data fetched from the models (via views) and allow user interaction via forms.

Typical templates include:

- Login and registration pages
- Account dashboard
- Transaction history page
- Admin views for user and transaction management

Templates support template inheritance and include modular components like headers, footers, and navigation bars for a consistent layout.

4. Admin Interface




Django's built-in admin interface provides a powerful backend for administrators. It enables staff to:

- Add, edit, or delete users and accounts
- View and filter transactions
- Monitor system activity and data integrity
- Perform administrative tasks without building custom views

The admin interface is secured via Django's authentication and permissions system and can be customized with model admin classes.

Key Design Considerations

To ensure security, maintainability, and scalability, several architectural best practices were implemented:

-  Use of Django's Authentication System
Django's robust authentication system provides secure session handling, password hashing, and built-in views for login, logout, and password management. It supports permission-based access and integrates seamlessly with both views and the admin panel.
-  Normalized Data Models
The database schema follows normalization principles to avoid data redundancy. Relationships are clearly defined (e.g., One-to-Many from User to Account, and Account to Transaction), which simplifies queries and data consistency.
-  Access Control in Views
Views that require user authentication are protected with `@login_required` decorators. Additionally, role-based permissions are enforced (e.g., only admin users can access certain views or modify sensitive records). This ensures that users can only access what they are authorized to.

➤ DATA STRUCTURE OF ALL MODULES:

```
mysql> show tables;
+-----+
| Tables_in_banking_system |
+-----+
| accounts_bankaccounttype |
| accounts_user             |
| accounts_user_groups      |
| accounts_user_user_permissions |
| accounts_useraddress      |
| accounts_userbankaccount  |
| auth_group                |
| auth_group_permissions    |
| auth_permission           |
| django_admin_log          |
| django_celery_beat_clockedschedule |
| django_celery_beat_crontabschedule |
| django_celery_beat_intervalschedule |
| django_celery_beat_periodictask |
| django_celery_beat_periodictasks |
| django_celery_beat_solarschedule |
| django_content_type       |
| django_migrations         |
| django_session            |
| transactions_transaction   |
+-----+
20 rows in set (0.02 sec)
```

```
mysql> show tables;
+-----+
| Tables_in_banking_system |
+-----+
| accounts_bankaccounttype |
| accounts_user             |
| accounts_user_groups      |
| accounts_user_user_permissions |
| accounts_useraddress      |
| accounts_userbankaccount  |
| auth_group                |
| auth_group_permissions    |
| auth_permission           |
| django_admin_log          |
| django_celery_beat_clockedschedule |
| django_celery_beat_crontabschedule |
| django_celery_beat_intervalschedule |
| django_celery_beat_periodictask |
| django_celery_beat_periodictasks |
| django_celery_beat_solarschedule |
| django_content_type       |
| django_migrations         |
| django_session            |
| transactions_transaction   |
+-----+
20 rows in set (0.02 sec)
```

```
mysql> select*from accounts_bankaccounttype;
```

id	name	maximum_withdrawal_amount	annual_interest_rate	interest_calculation_per_year
1	saving	100000.00	10.00	5
2	Current Account	100000.00	0.50	12
3	Fix Deposit	10000.00	6.00	12

```
3 rows in set (0.00 sec)
```

```
mysql> select*from accounts_user;
```

id	password	superuser	first_name	last_name	is_staff	is_active	date_joined	email	last_login	is_
1	pbkdf2_sha256\$260000\$a8xkdjIqelc0ufJqq0U4dq\$klGveBk0hbAHR+CwtvUc/+zjl7B6HekMkCqbArp8Wq8=	1			1	1	2025-05-10 09:14:21.000244	admin@gmail.com	2025-05-25 19:01:08.767296	
2	1234	0	satyam	gopale	0	1	2025-05-10 09:17:46.000000	Satyamgopale17@gmail.com	2025-05-14 09:18:19.000000	
3	pbkdf2_sha256\$870000\$qVv0G66BPS73u2LzkzLqx0\$3es3cZc1ne3gv9jubGqrc0C7hIw8IwInkhmRLY2U2RY=	0	satyam	gopale	0	1	2025-05-10 09:21:57.580413	satyamgopale@gmail.com	2025-05-10 09:21:58.765485	
4	pbkdf2_sha256\$870000\$U8xdaSG7iHA7cLGJ1NyhUK\$rX79Ld8+JTp0VvH38e7KNLDnhvkHkbLsMMBOMl3kvxU=	0	satyam	gopale	0	1	2025-05-22 05:10:47.401015	Satyamgopale474@gmail.com	2025-05-22 05:10:48.215901	
5	pbkdf2_sha256\$1000000\$EarXMUQHgNyAPN0Kko4nq\$WBPkNVJNfLHmzjrw4ccpVQnLhn4FDtZyPEnS00ZoXXk=	0	Satyam	Gopale	0	1	2025-05-25 18:32:13.672949	SS0080348600@gmail.com	2025-05-26 07:16:32.912550	

```
5 rows in set (0.00 sec)
```

```
mysql> select*from accounts_user_groups;  
Empty set (0.03 sec)
```

```
mysql> select*from accounts_user_user_permissions;  
Empty set (0.00 sec)
```

```
mysql> select*from accounts_useraddress;
```

id	street_address	city	postal_code	country	user_id
1	.	.	422602	India	3
2	.	.	422602	India	4
3	pimple Gurav	pune	422602	India	5

```
3 rows in set (0.02 sec)
```

```
mysql> select*from accounts_userbankaccount;
```

id	account_no	gender	birth_date	balance	interest_start_date	initial_deposit_date	account_type_id	user_id
1	1000000003	M	2003-01-17	15000.00	2025-07-10	2025-05-10	1	3
2	1000000004	M	2003-01-17	5300.00	2025-07-22	2025-05-22	1	4
3	1000000005	M	2003-01-17	445800.00	2025-07-25	2025-05-25	1	5

```
3 rows in set (0.01 sec)
```

```
mysql> select*from auth_group;  
Empty set (0.01 sec)
```

```
mysql> select*from auth_group_permissions;  
Empty set (0.01 sec)
```



```
mysql> select*from auth_group_permissions;
Empty set (0.01 sec)

mysql> select*from django_admin_log;
+-----+-----+-----+-----+-----+-----+-----+-----+
--+
| id | action_time          | object_id | object_repr          | action_flag | change_message | content_type_id | user_id |
+-----+-----+-----+-----+-----+-----+-----+-----+
--+
| 1 | 2025-05-10 09:18:23.865916 | 2 | Satyamgopale17@gmail.com | 1 | [{"added": {}}] | 12 | 1 |
| 2 | 2025-05-10 09:19:11.871334 | 1 | saving | 1 | [{"added": {}}] | 13 | 1 |
| 3 | 2025-05-25 18:54:01.731026 | 2 | Current Account | 1 | [{"added": {}}] | 13 | 1 |
| 4 | 2025-05-25 19:00:14.342310 | 3 | Fix Deposit | 1 | [{"added": {}}] | 13 | 1 |
+-----+-----+-----+-----+-----+-----+-----+-----+
--+
4 rows in set (0.02 sec)

mysql> select*from django_celery_beat_clockedschedule;
Empty set (0.02 sec)

mysql> select*from django_celery_beat_crontabschedule;
Empty set (0.02 sec)

mysql> select*from django_celery_beat_intervalschedule;
Empty set (0.02 sec)

mysql> select*from django_celery_beat_periodictask;
```

```
mysql> select*from django_celery_beat_periodictasks;
Empty set (0.02 sec)



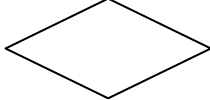

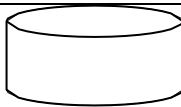
mysql> select*from django_celery_beat_solarschedule;
Empty set (0.02 sec)

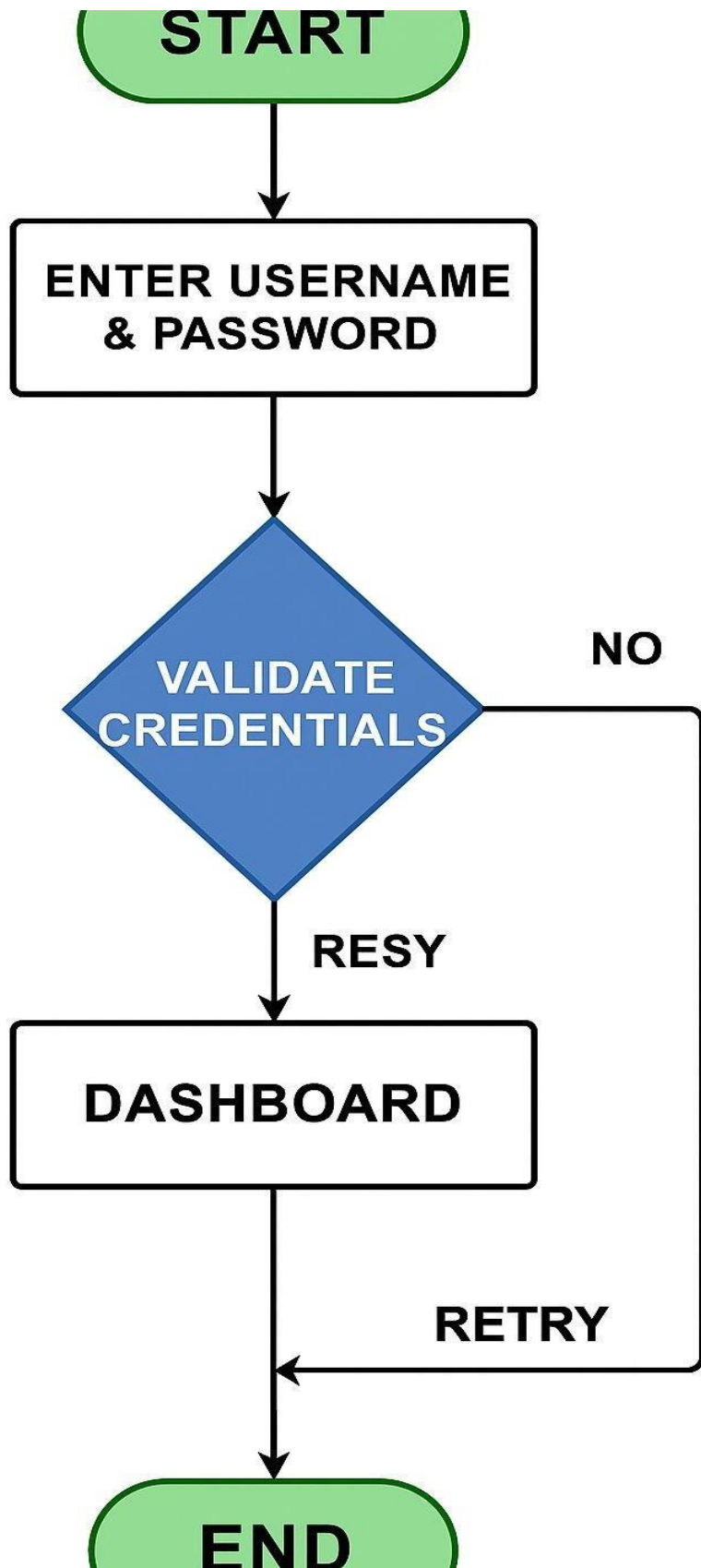
mysql> select*from django_content_type;
+-----+-----+-----+
| id | app_label | model |
+-----+-----+-----+
| 13 | accounts | bankaccounttype |
| 12 | accounts | user |
| 15 | accounts | useraddress |
| 14 | accounts | userbankaccount |
| 1 | admin | logentry |
| 3 | auth | group |
| 2 | auth | permission |
| 4 | contenttypes | contenttype |
| 11 | django_celery_beat | clockedschedule |
| 6 | django_celery_beat | crontabschedule |
| 7 | django_celery_beat | intervalschedule |
| 8 | django_celery_beat | periodictask |
| 9 | django_celery_beat | periodictasks |
| 10 | django_celery_beat | solarschedule |
| 5 | sessions | session |
| 16 | transactions | transaction |
+-----+-----+-----+
16 rows in set (0.01 sec)

mysql> select*from django_migrations;
```

PROCEDURAL DESIGN:

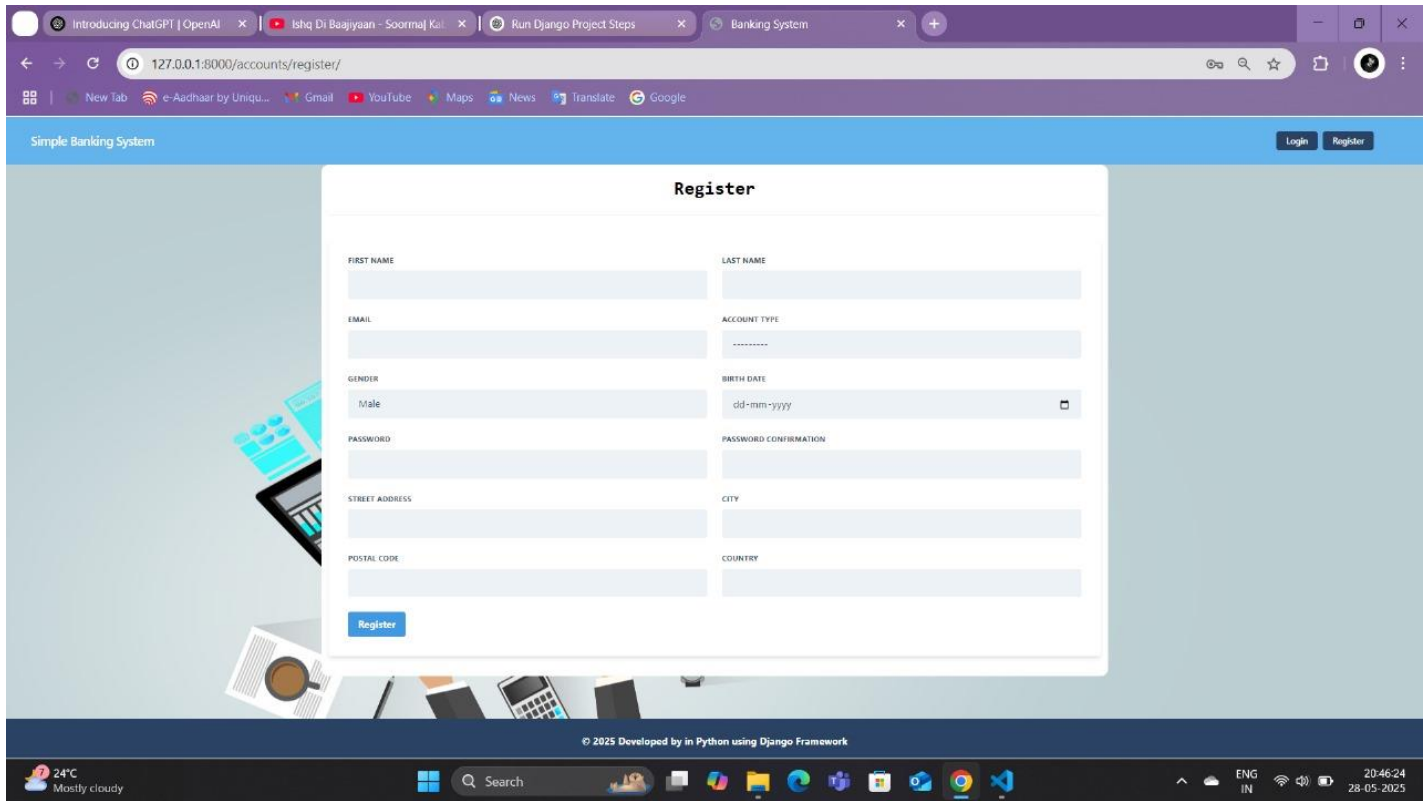
PROCESS LOGIC (FLOW CHART) OF EACH MODULE

<u>Notations Used For FLOW CHART</u>		
NOTATIONS		USED FOR
	Known as TERMINATOR	Used for START / STOP
	Known as DATA	Used for ACCEPTING INPUTS FROM USER AND also DISPLAYING THE ERROR MESSAGES GENERATED BY THE SYSTEM
	Known as DECISION	Used for DECISION MAKING
	Known as DISPLAY	Used for OUTPUT GENERATED BY THE SYSTEM
	Known as MAGNETIC DISK	Used for STORING INPUTS GIVEN TO THE SYSTEM



INPUT SCREENS:

Register page:



Simple Banking System

127.0.0.1:8000/accounts/register/

Register

First Name:

Last Name:

Email:

Account Type:

Gender:

Birth Date:

Password:

Password Confirmation:

Street Address:

City:

Postal Code:

Country:

Register

© 2025 Developed by in Python using Django Framework

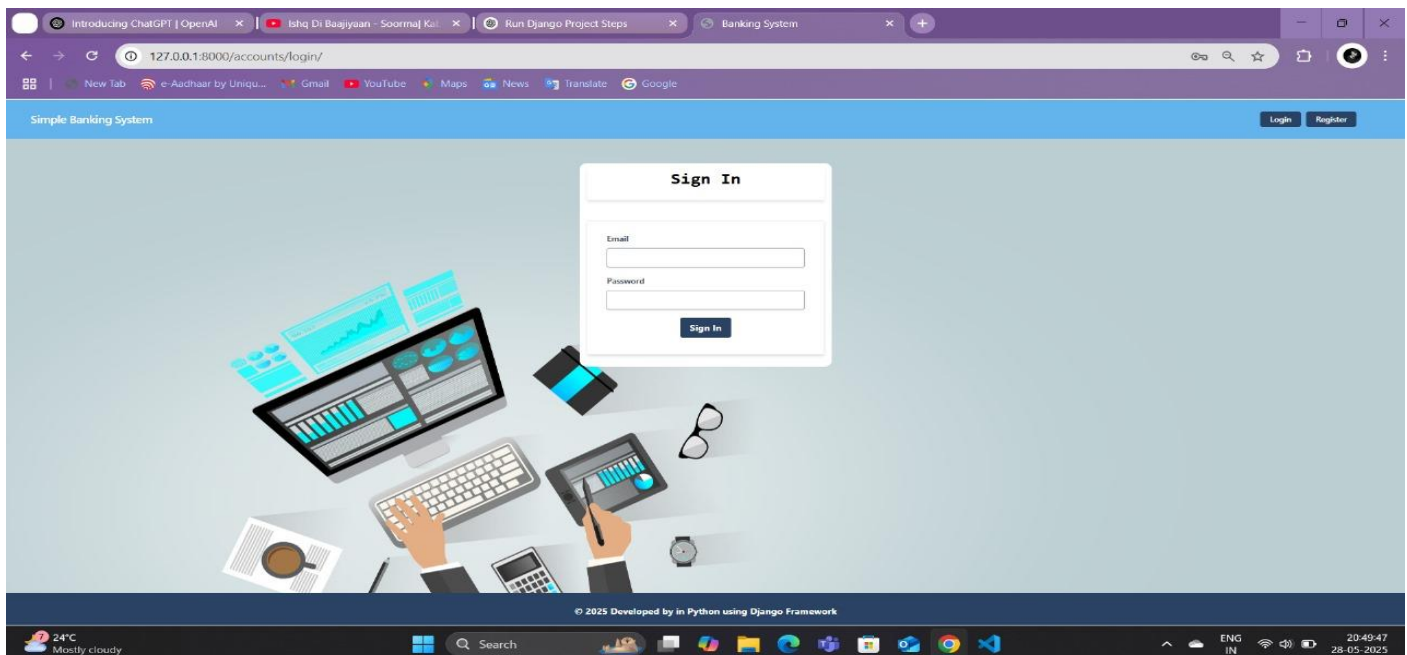
24°C Mostly cloudy

Search

ENG IN

20:46:24 28-05-2025

Sign Page:



Simple Banking System

127.0.0.1:8000/accounts/login/

Sign In

Email:

Password:

Sign In

© 2025 Developed by in Python using Django Framework

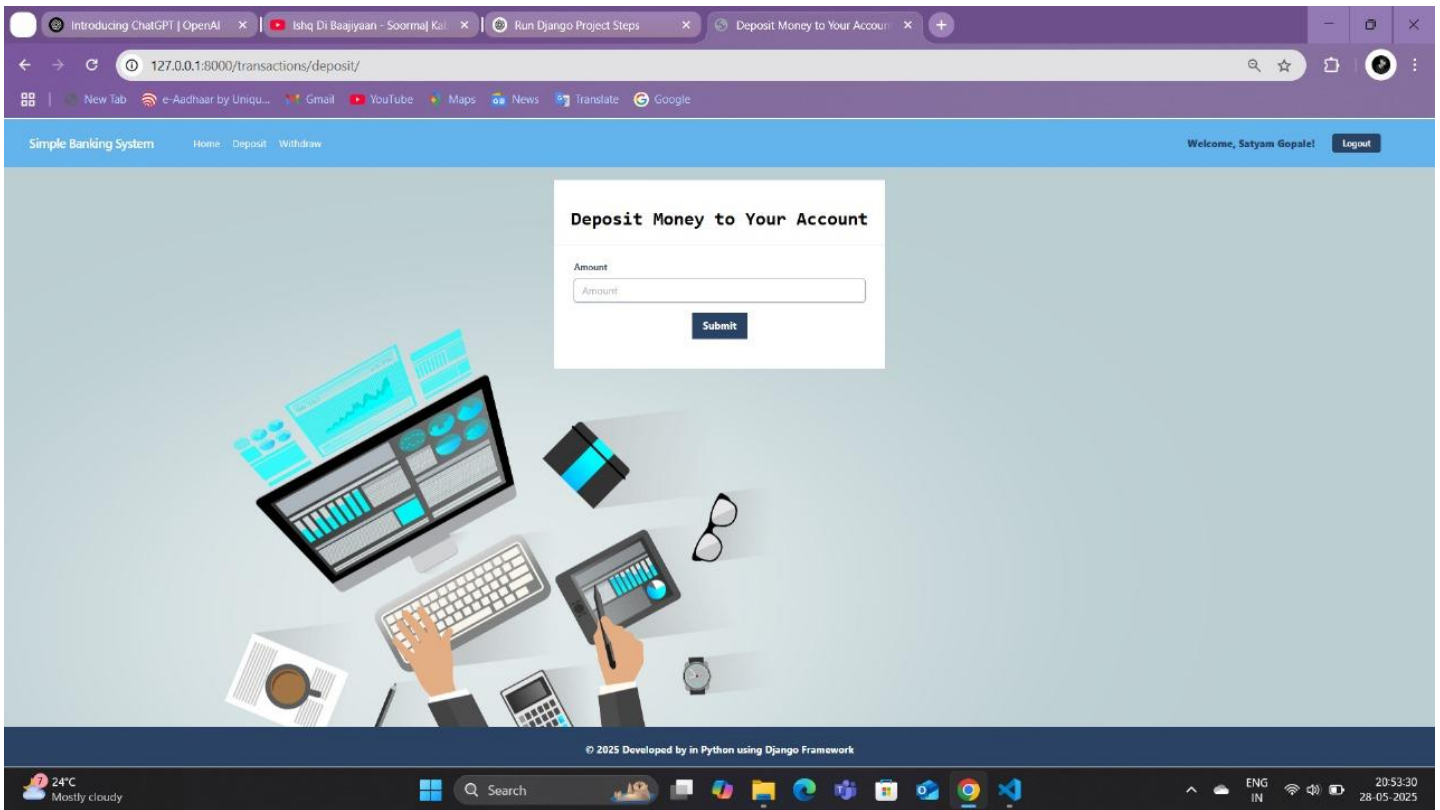
24°C Mostly cloudy

Search

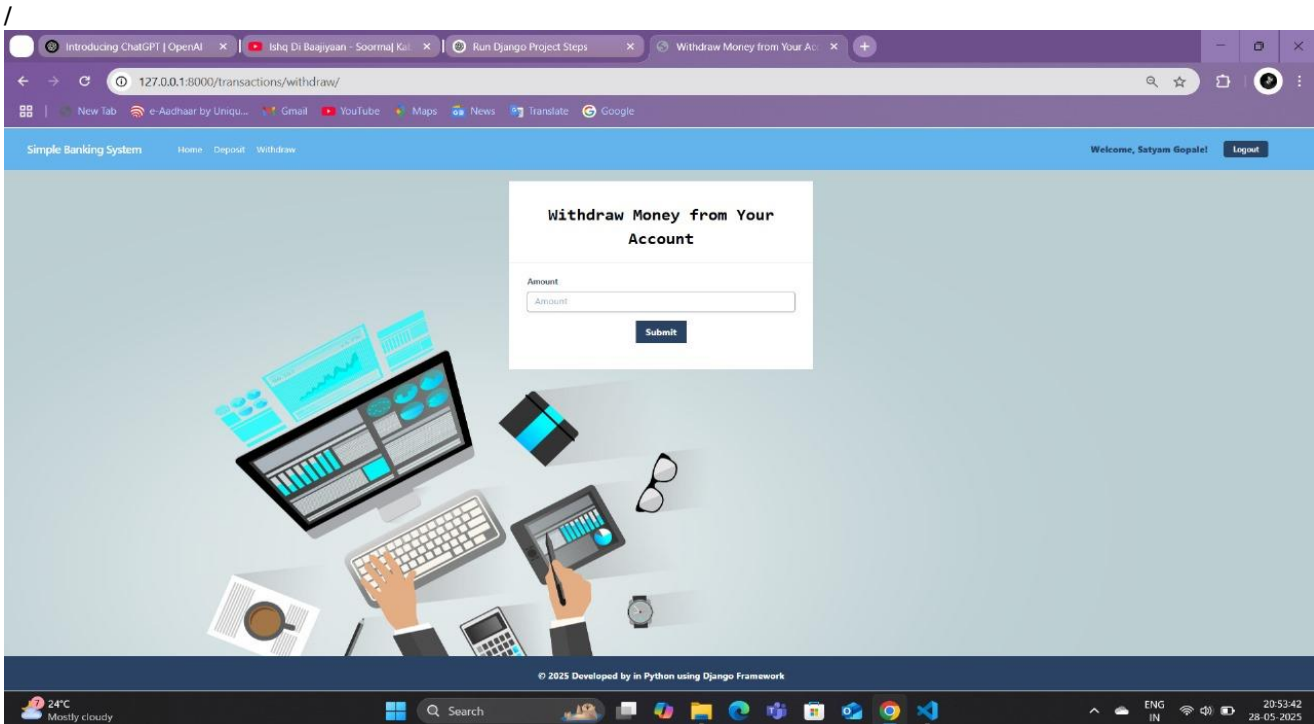
ENG IN

20:49:47 28-05-2025

Deposit page:



Withdraw page:



LIST OF REPORTS THAT ARE LIKELY TO BE GENERATED:

1. Registration Page:

This page allows users to create an account by filling in personal and banking details like name, gender, address, account type, and password, ensuring secure onboarding for new customers.

2. Home Page:

The homepage displays basic navigation options like transaction reports, deposits, and withdrawals. It features a minimal design, allowing users to easily register or log in for online banking access.

3. Deposit Page:

This page provides users with a simple form to deposit money into their account. It includes an amount field and a submit button for initiating the deposit transaction securely.

4. Withdraw Page:

Users can withdraw funds from their account by entering the desired amount. The interface is clean and consistent with the rest of the banking system, ensuring ease of use.

➤ ERROR HANDLING:

Errors are indicators of issues within the project implementation. A project is considered successful only when it runs with minimal or no errors. Hence, identifying and addressing errors is a critical part of developing a Hotel Management System using Django. Errors encountered during the development and deployment of this project can be classified into the following categories:

Syntax Errors:

These errors occur when the developer violates the grammatical rules of Python or Django syntax. For instance, missing colons, incorrect indentation, or improper use of Python or Django-specific commands (e.g., incorrect model or view syntax).

Such errors are caught at the time of code writing or compilation.

Logical Errors:

Logical errors arise when the system executes without crashing but produces incorrect or unintended output.

These errors are not flagged by the Django framework or Python interpreter and can only be found through rigorous testing of application functionality. Thorough testing of views, forms, and templates is essential to uncover and fix logical flaws in the system.

Runtime Errors:

These are the most complex errors to detect as they occur only during the execution of the application. These could be caused by unexpected user input, server-side exceptions, or environmental issues such as database connection failures.

In Django, runtime errors can include issues like:

- Form validation failures due to missing or incorrect user input.

➤ **VALIDATION CHECKS**

Software testing is a vital part of developing a robust Hotel Management System. It often falls under the broader concept of **Verification and Validation (V & V)**.

- **Verification** ensures that the system is built according to the specifications and performs the intended functions correctly.
- **Validation** ensures the product meets the business needs and customer requirements.

Boehm's states this another way:

- **Verification:** “Are we building the product right?”
- **Validation:** “Are we building the right product?”

The concept of V & V includes several practices that contribute to Software Quality Assurance (SQA), which is integral to any Hotel Management System.

Validation in Django Forms

Validation is commonly handled in Django using **forms** and **model field validators**. In this project, we use the following types of validation:

- Empty Field Validation
- Numeric Field Validation



TESTING STRATEGY

Testing is an essential phase in the development of the **Hotel Management System using Django**, where the primary goal is to ensure that each module—such as room booking, guest management, payment handling, and admin panel—works as intended.

Testing provides a clear, unbiased view of the system's reliability and performance and helps verify that the software meets business requirements and user expectations.

The objective is to:

1. Ensure the system meets hotel business and technical requirements.
2. Confirm all modules perform as expected.
3. Guarantee consistent and reproducible performance across similar use cases.

Software testing methods applied during the Django project development include:

Unit Testing

Unit testing is used to test individual modules in isolation—like the `booking()`, `check_availability()`, and `calculate_total()` functions in the backend views.

- Each function is tested to ensure it returns the correct response, handles exceptions, and validates user input properly.
- Django's built-in Test Case class is used for writing these tests.
- This ensures each core logic block, like calculating room rates or checking availability, works independently.

Example:

Test that the booking function rejects a request when a room is already occupied during the requested date range.

Integration Testing

Integration testing validates how modules interact—like how the booking page communicates with the database and payment gateway.

- Modules like the frontend room booking form, backend processing logic, and database models are integrated and tested.
- This ensures that data flows correctly from form input through Django views to the database.
- Django's testing framework and tools like `pytest-django` or Selenium (for UI flows) are used.

Example:

Test the interaction between the user room booking form, room model, and payment module.

System Testing

System testing checks the entire hotel management system to ensure it behaves correctly as a whole.

- This includes testing all features end-to-end: user login, room search, booking confirmation, and payment generation.
- Tests are conducted to ensure outputs match expected outcomes from real user scenarios.

Example:

Create a test user, simulate a complete room booking from login to confirmation, and verify system responses at each step.



Security is one of the most critical aspects of any banking or financial software system. The Banking System developed using Python, Django, and MySQL has been designed with a multi-layered security approach to ensure the protection of user data, transaction integrity, and system reliability. This section outlines the key security strategies and implementations integrated into the system.

1. User Authentication & Password Protection

The system uses Django's built-in authentication framework to manage user logins and sessions.

- * Passwords are never stored in plain text; Django hashes them using the PBKDF2 algorithm with a salt, making them resilient against brute-force and rainbow table attacks.
- * Login views are protected from repeated login attempts by implementing rate limiting or by integrating third-party packages such as django-axes or django-ratelimit.

2. Role-Based Access Control

- * There are two primary user roles: Customer and Admin.
- * Role-specific permissions are enforced using decorators like `@login_required` and `@user_passes_test`.
- * Admin functionalities such as managing user accounts or viewing transaction logs are restricted to users with `is_staff` or `is_superuser` privileges.
- * Django's admin panel is secured with an authentication gate and CSRF protection.

3. Session Management

- * Django uses secure session IDs stored in cookies that are encrypted and tied to the user's browser.
- * The `SESSION_COOKIE_SECURE` flag is enabled in production environments to ensure that session cookies are only transmitted over HTTPS.
- * Session timeouts are configured to automatically log users out after a period of inactivity.

4. Cross-Site Request Forgery (CSRF) Protection

- * All forms and POST requests are protected with CSRF tokens provided automatically by Django's middleware.
- * Any request without a valid CSRF token is rejected, protecting users from malicious site submissions.

5. SQL Injection Prevention

- * Django ORM handles all database queries through safe and parameterized queries.
- * Direct use of raw SQL is avoided unless absolutely necessary, and even then, parameters are safely escaped using Django's raw query utilities.
- * Input validation is enforced at both form and model levels.



Report Page:

Simple Banking System Home Deposit Withdraw Welcome, Satyam Gopale! Logout

Transaction Report

Filter using date range

DATE	TRANSACTION TYPE	AMOUNT	BALANCE AFTER TRANSACTION
May 28, 2025 11:22 PM	Deposit	\$ 5,000.00	\$ 5,000.00
May 28, 2025 11:22 PM	Withdrawal	\$ 200.00	\$ 4,800.00
May 28, 2025 11:22 PM	Deposit	\$ 6,000.00	\$ 10,800.00
May 28, 2025 11:22 PM	Withdrawal	\$ 800.00	\$ 10,000.00
Current Balance			\$ 10,000.00

© 2025 Developed by in Python using Django Framework

24°C Mostly cloudy Search ENG IN 20:53:18 28-05-2025



FUTUR SCOPE OF THE PROJECT

The current version of the Banking System provides essential functionalities such as account management, transactions, and administrative monitoring using Python, Django, and MySQL. While it offers a secure and scalable foundation for digital banking operations, there is significant potential for future enhancements and extensions. The system can be further evolved in the following areas:

Mobile Banking App

Developing a native or cross-platform mobile application (using Flutter, React Native, or Kotlin/Swift) will provide customers with greater convenience and flexibility to manage their accounts on the go. The mobile app can mirror the core features of the web system and introduce additional mobile-centric features such as biometric login (fingerprint/face recognition).

Two-Factor Authentication (2FA)

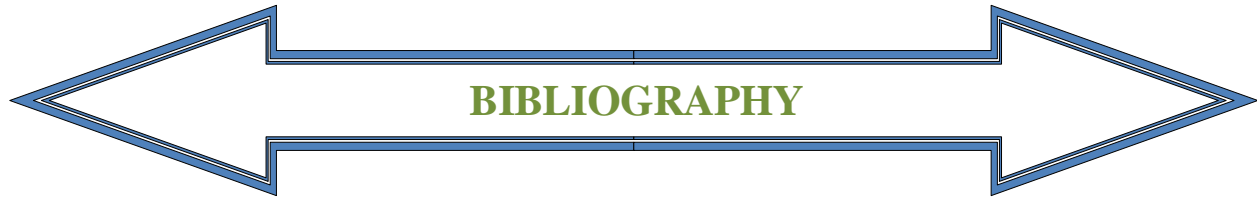
To enhance login security, 2FA can be implemented using OTPs (One-Time Passwords) sent via SMS or email, or through integration with authenticator apps like Google Authenticator or Authy. This would add an extra layer of protection for sensitive operations like login, fund transfer, or password reset.

Credit and Loan Management

A module for managing credit cards, EMI-based loans, and credit scoring can be added. Users could apply for loans, track repayment schedules, and view eligibility metrics. Admins could review and approve loan requests with automated calculations and risk evaluation.

AI-Based Fraud Detection

Incorporating machine learning algorithms can help monitor user behavior and detect anomalies or suspicious transactions. For example, if a user logs in from a different country or makes large, unusual transfers, the system could trigger alerts or temporarily block the action.



1. Django Software Foundation. (n.d.). *Django Documentation*. Retrieved from <https://docs.djangoproject.com>
2. Python Software Foundation. (n.d.). *Official Python Documentation*. Retrieved from <https://docs.python.org>
3. W3Schools. (n.d.). *HTML, CSS, and JavaScript Tutorials*. Retrieved from <https://www.w3schools.com>
4. Geeks for Geeks. (n.d.). *Django Tutorials and Examples*. Retrieved from <https://www.geeksforgeeks.org>
5. Tutorialspoint. (n.d.). *Django and Python Programming Guides*. Retrieved from <https://www.tutorialspoint.com>
6. Stack Overflow. (n.d.). *Community Discussions and Developer Solutions*. Retrieved from <https://stackoverflow.com>
7. MDN Web Docs. (n.d.). *HTML and CSS Reference*. Retrieved from <https://developer.mozilla.org>
8. MySQL Documentation. (n.d.). *MySQL 8.0 Reference Manual*. Retrieved from <https://dev.mysql.com/doc>
9. Real Python. (n.d.). *Practical Django Projects and Python Tips*. Retrieved from <https://realpython.com>