CERTIFICATE

This is to certify that the report on "**Automated Teller Machine (ATM) System**" is a bonafide record of Project presented by **Kundan Bhasin** (Roll.no:1748710007) towards partial fulfillment for the award of the degree in B.Tech. computer Science by the University of AKTU.

Staff in Charge   Head of the Department

Place: ………………..

Date: ....

DECLARATION

Kundan Bhasin hereby declare that the project work entitled "**Automated Teller Machine (ATM) System**" submitted to aktu. University in partial fulfillment of the requirements for the award of degree of B.Tech Computer Science is a record of the original project work done during the period of 26th Feb 2021 in PIET College For Advanced Studies.

Place: ………………..   Sign of Candidate

Date: ....

ACKNOWLEDGEMENT

I extend my sincere gratitude toward**s Prof. Gaurav Pardhan** Head of Department for giving us his invaluable knowledge and wonderful technical guidance.

I express my thanks to Mr**. Gaurav Pardha**n , our Computer Lab Instructor and also to our staff advisor Ms. **Narender Sing**h for their kind cooperation and guidance for preparing and presenting this seminar.

I also thank all the other faculty members of B.Tech. Computer Science department and my friends for their help and support.

ABSTRACT

An Automated Teller Machine (ATM) is a safety-critical and real-time system that is highly complicated in design and implementation. This paper presents the formal design, specification, and modeling of the ATM system using a denotational mathematics known as Real-Time Process Algebra
(RTPA). The conceptual model of the ATM system is introduced as the initial requirements for the system.

The architectural model of the ATM system is created using RTPA architectural modeling methodologies and refined by a set of Unified Data Models (UDMs), which share a generic mathematical model of tuples. The static behaviors of the ATM system are specified and refined by a set of Unified Process Models (UPMs) for the ATM transition processing and system supporting processes. The dynamic behaviors of the ATM system are specified and refined by process priority allocation, process deployment, and process dispatch models. Based on the formal design models of the ATM system, code can be automatically generated using the RTPA Code Generator (RTPA-CG), or be seamlessly transformed into programs by programmers. The formal models of ATM may not only serve as a formal design paradigm of real-time software systems, but also a test bench for the expressive power and modeling capability of exiting formal methods in software engineering.

INDEX

CHAPTER 1

INTRODUCTION

1.INTRODUCTION:

We are very glad to introduce our project "**AUTOMATED TELLER MACHINE**". Nowadays each company or organization prefers computerized paper-work. Definitely the computer system is more reliable than the manual works. The common human errors can be eliminated with the help of a system.

An **Automated Teller Machine** (**ATM**) is a computerized telecommunications device that provides the customers of a financial institution with access to financial transactions in a public space without the need for a human clerk or bank teller. On most modern ATMs, the customer is identified by inserting a plastic ATM card with a magnetic stripe or a plastic smartcard with a chip that contains a unique card number and some security information.

ATMs are known by various other names including automated banking machine, money machine, bank machine, cash machine and Any Time Money in India.

An ATM card (also known as a bank card, client card, key card or cash card) is an ISO 7810 card issued by a bank, credit union or building society, Unlike a debit card, in-store purchases or refunds with an ATM card can generally be made in person only, as they require authentication through a personal identification number (PIN). In other words, ATM cards cannot be used at merchants that only accept credit cards.

What Is an Automated Teller Machine (ATM)?

An automated teller machine (ATM) is an electronic banking outlet that allows customers to complete basic transactions without the aid of a branch representative or teller. Anyone with a credit card or debit card can access cash at most ATMs.

ATMs are convenient, allowing consumers to perform quick self-service transactions such as deposits, cash withdrawals, bill payments, and transfers between accounts. Fees are commonly charged for cash withdrawals by the bank where the account is located, by the operator of the ATM, or by both. Some or all of these fees can be avoided by using

an ATM operated directly by the bank that holds the account.
ATMs are known in different parts of the world as automated bank machines (ABM) or cash machines.

Understanding Automated Teller Machines (ATMs) :

The first ATM appeared at a branch of Barclay's Bank in London in 1967, although there are reports of a cash dispenser in use in Japan in the mid-1960s. The interbank communications networks that allowed a consumer to use one bank's card at another bank's ATM came later, in the 1970s.

Within a few years, ATMs had spread around the globe, securing a presence in every major country. They now can be found even in tiny island nations such as Kiribati and the Federated States of Micronesia.

KEY TAKEAWAYS

- Automated teller machines are electronic banking outlets that allow people to complete transactions without going into a branch of their bank.
- Some are simple cash dispensers while others allow a variety of transactions such as check deposits, balance transfers, and bill payments.
- To keep ATM fees down, use an ATM branded by your own bank as often as possible.

Types of ATMs :

There are two primary types of ATMs. Basic units only allow customers to withdraw cash and receive updated account balances. The more complex machines accept deposits, facilitate line-of-credit payments and transfers, and access account information.

To access the advanced features of the complex units, a user must be an account holder at the bank that operates the machine.

To access the advanced features of the complex units, a user must be an account holder at the bank that operates the machine.
Analysts anticipate ATMs will become even more popular and forecast an increase in the number of ATM withdrawals. ATMs of the future are likely to be full-service terminals instead of or in addition to traditional bank tellers.

The average amount of cash withdrawn from an ATM per transaction.

Although the design of each ATM is different, they all contain the same basic parts:

- **Card reader**: This part reads the chip on the front of the card or the magnetic stripe on the back of the card.
- **Keypad**: The keypad is used by the customer to input information, including personal identification number (PIN), the type of transaction required, and the amount of the transaction.
- **Cash dispenser**: Bills are dispensed through a slot in the machine, which is connected to a safe at the bottom of the machine.

- **Printer**: If required, consumers can request receipts that are printed here. The receipt records the type of transaction, the amount, and the account balance.
- **Screen**: The ATM issues prompts that guide the consumer through the process of executing the transaction. Information is also transmitted on the screen, such as account information and balances.

Full-service machines now often have slots for depositing paper checks.

Special Considerations: Using ATMs :

Banks place ATMs inside and outside of their branches. Other ATMs are located in high traffic areas such as shopping centers, grocery stores, convenience stores, airports, bus and railway stations, gas stations, casinos, restaurants, and other locations. Most ATMs that are found in banks are multi-functional, while others that are offsite tend to be primarily or entirely designed for cash withdrawals.

ATMs require consumers to use a plastic card—either a bank debit card or a credit card—to complete a transaction. Consumers are authenticated by a PIN before any transaction can be made.

Many cards come with a chip, which transmits data from the card to the machine. These work in the same fashion as a bar code that is scanned by a code reader.

ATM Fees:

Account-holders can use their bank's ATMs at no charge, but accessing funds through a unit owned by a competing bank usually incurs a fee. According to MoneyRates.com, the average fee to withdraw cash from an out-of-network ATM was $4.61 as of late 2019.

Some banks will reimburse their customers for the fee, especially if there is no corresponding ATM available in the area.

So, If you're one of those people who draws weekly spending money from an ATM, using the wrong machine could cost you nearly $240 a year.

CHAPTER 2

Existing System and need for the new system

Existing System:-

- Existing system creates complexity in doing calculation of collection of money because it is manual.
- Existing system creates problems in maintaining records of book keeping.
- Existing system has large documentary work so it requires space for its storage.
- To do the documentary work there is a need of extra staff worker.
- Existing system takes much more time to update the process of records.
- Due to the existing system, crowding of customers in bank premises are more & pressure of work on bank servants are also more.

Need for the new system:-

- The proposed system need to maintain all the records in computerized form.
- It is useful to store record systematically & accurately by using this system.
- It is useful to reducing the extra work which maintains the records of book keeping & paper less work.
- We can easily handle data efficiently & effectively.
- The storage space, extra workers, missing files all these possibilities are decreased through this system.
- This system helps to save time & cost spending on documentation.
- With the help of this system, an ATM card holder can see all the records about his account only at any time efficiently.

- The most important facility provided by this system is that, there is no possibility of miss any records.

- This system is useful for recording daily transactions done by customers.

- So, this system helps to overcome the problems of previous system.

- An ATM card is an ISO card issued by a bank, credit union or building society. Unlike a debit card, in-store purchases or refunds with an ATM card can generally be made in person only, as they require authentication through a personal identification number or PIN. In other words, ATM cards cannot be used at merchants that only accept credit cards.

CHAPTER 3

SCOPE OF THE WORK

Scope of the work:-

As this is software it can be used by a wide variety of banks to automate the process of manually maintaining the records related to the each transaction of bank account holder. The main goal of this application is to provide very reliable & efficient service to bank account holder at any time & any location.

This system will cover the following modules,

- Cash Withdrawal.
- Balance Enquiry.
- Mini Statement.
- PIN Change.
- Cash Deposit.
- Loan Information.
- Help Menu.

Detail information about system modules,

- **Cash Withdrawal :-** It is mainly used for withdrawal of cash as per customer demand. For any authorized ATM card holder the ATM system requests for its ATM no & PIN no then customer to login in their accounts, then amounts are given to the system and customer can withdraw amount.
- **Balance Enquiry :-** It refers to enquiry of bank balance of an authorized ATM card holder account to check for the resulting balance after certain transactions.

- **Mini Statement:-** It refers to enquiry of the last ten transactions of an authorized ATM card holder. It includes deposit & withdrawal amount of transaction & also contains respective transaction date and current available balance.

- **PIN Change:-** It refers to the Change of PIN no of an authorized ATM card holder. I require giving the system old PIN no of the ATM card & then giving new PIN no & confirm the new PIN no.

- **Cash Deposit:-** It is mainly used for deposit cash amounts to their bank account as per customer demand. It is easy process to deposit amount to their bank accounts without filling deposit sleep.

- **Loan Information:-** It mainly used for to give information about various bank loan rates to ATM card holder.

- **Help Menu:-** Help menu gives information about how to use ATM card when you give correct information to ATM system. I describe information about all transaction menus & what is used of each menu.

Feasibility study:-

- Feasibility Study is essential to evaluate cost & benefit of the proposed system. This is very important step because on the basis of this; system decision is taken on whether to proceed or to postpone the project or to cancel the project. Feasibility study forms the most important phase in the system development life cycle so that the people who are affected by the system benefit from the change. This involves some very crude estimates of schedules of completion of the proposed system and the cost of the system. This study ensures that the system meets the objectives of the organization before it can be approved for development. It also involves the study of different risks involved in developing the system**.**

The major areas to consider while determining the feasibility of a system are:-

- **Technical Feasibility:-**

The technical feasibility study always focuses on the existing computer hardware, software and personal. This also includes need for more hardware, software or personal and possibility of procuring or installing such facilities.

ATM is a system that can work on single stand alone Pentium machine with 128 MB RAM, Hard disk drive size of 80 GB, mouse, monitor and keyboard & it also require internet connection to corresponding computer. The equipments are easily available in the market, so technically the system is very much feasible.

- **Economical Feasibility: -** This feasibility is useful to find the system development cost and checks whether it is justifiable. The cost overheads include software and hardware maintenance cost, training costs that includes cost required for manpower, electricity, stationary etc. The proposed system will provide the right type of information at right time, and in the required format. This will save time required for decision-making and routine operations. Considering all these advantages, the cost overheads of the system are negligible. So the system is economically feasible.

- **Operational Feasibility : -** It is also known as resource feasibility. The operation users of the system are expected to have minimum knowledge of computer. The developed system is simple to use, so that the user will be ready to operate the system. The proposed system is developed using JAVA programming language & Mysql database which is platform independent and user friendly. So the system is operationally feasible.

Hardware Requirements:-

- Processor: Pentium 4 or onwards.
- Hard Disc: 80GB.
- RAM: 128MB.
- Monitor: 15” Color Monitor.
- Mouse.
- Keyboard.

Software Requirements:-

Operating System: - Windows 10 or onwards or Linux.

- Python RunTime Environment (JRE) – jdk1.5 (As Front End Tool).
- Java script-connector-python-9.1.2-bin.
- JS server (As Back End Tool).
- JCreator.

Programming Languages Used:-

In this system we use JAVA Platform for programming language.JAVA Platform means the environment which is used to run program.JAVA is platform independent language since no only single operating system can be required by the java. All the different operating system can execute the java programming language.

Java provides huge functionality that means it provide A huge library.

- Containing lots of reusable codes.
- An execution environment that provides services such as security.
- Portability across operating system.
- Automatic garbage collection.

Requirement Analysis:-

This involves studying the current system to find out how it is working and where the improvements should be made. These studies consider both manual and computer methods. Hence an early step in investigation is to understand situation.

Activities In Requirement Determination:

- Requirement Investigation:
- Requirement Specification:

- ### **Requirement Investigation**:-

This activity is at the heart of system analysis. Using a variety of tools and skills analyst study the current system and documents its features for further analysis. Requirement investigation relies on the fact-finding techniques

- **Requirement Specification**:-

The data produced during fact-finding investigation are analyzed to determine requirement specification. This is the description of features for new system.

Fact Finding Techniques:-

Fact-Finding is the formal process of using research, interviews, questionnaires, sampling and preferences. It is also called information about systems, requirements, and preferences. It is also called **Information Gathering** or **Data Collection**. Tools, such as data and process models, document facts, and conclusion are drawn from facts. If you can't collect the facts, you can't use the tools. Fact-Finding skills must be learned and practiced.

Different types of Fact-Finding techniques are:

* INTERVIEWS
* QUESTIONNAIRE
* RECORD REVIEW
* OBSERVATION

- **Interviews :**

Interview technique is used to collect the information from individuals Analyst should select responds that are related with the system under study. In this method the interviewer (analyst) faces to face with respondent & records of his/her responses. This interviewer must plane in advance and should fully know the problems under consideration. He must choose a suitable time & place, so that the interviewer may feel at ease during interview.

- **Questionnaire:**

A questionnaire performs containing a sequence of questions to elicit information mostly from a large no of persons. Drafting of questionnaires requires skill. The questions must be clear, simple & to the point. They must be well organized from the point of view of the respondent and formulated in such a manner as to provide the data in so far as possible in the desired form. A questionnaire may be mailed to individuals who are requested to write the answer of each question and return complemented performs back by post.

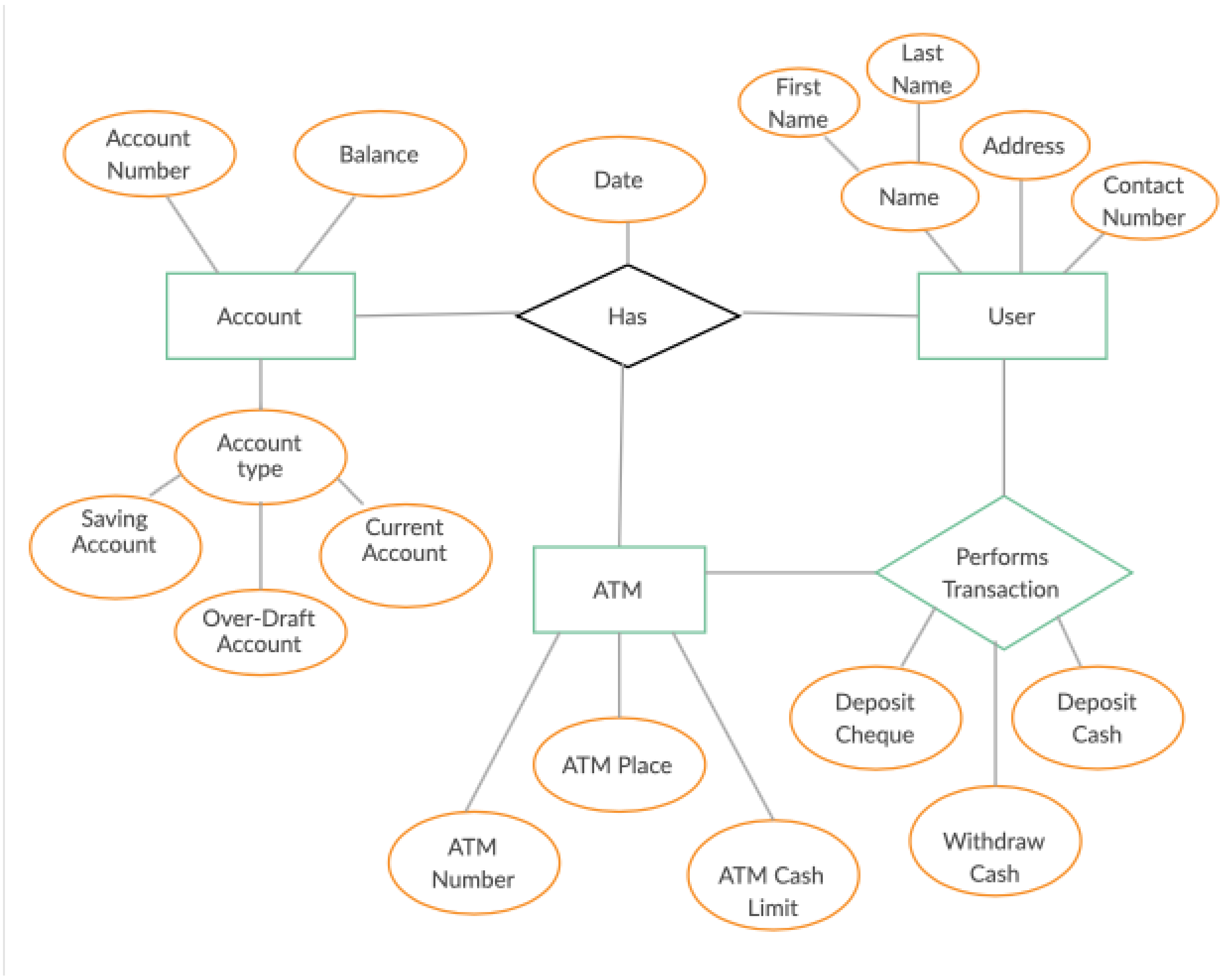
- **Record view:**

Information related with the system may be present in the form of records like books, magazines, newspaper, historical documents, letters, journals, manuals, government publications. This kind of record review provides very valuable information to the analyst about the system, organization & various procedures & rules.
Record review may be performed in the beginning of study to collect initial information or at the end of the study to compare actual operations.

- **Observations:**

If information is not collected from the other fact-finding methods, then observation method is used. In this method analyst observes the flow of documents, way the process is carried out, step followed, the persons involved etc. If the analyst is familiar with the system then he/she knows what to observe and how to gather information. In experienced person may observe unnecessary things, which delays the system study.

Entity Relationship Diagram:-



Normalized Database Design & Data Dictionary:-

Database Table Design:- The general theme behind a database is to handle information as an integrated whole. A database is a collection of interrelated data stored with minimum redundancy to serve many users quickly and effectively. After designing the input and output, the analyst must concentrate on database design or how data should be organized around user requirements. The general objective is to make information access, easy quick, inexpensive and flexible for other users. During database design the following objectives are concerned:-

- Controlled Redundancy
- Easy to learn and use
- More information and low cost
- Accuracy
- Integrity

Table Name: -

account detail :-

It stores the information about account detail of ATM card holder.

Field name	**Data type**	**Size**	**Constraint**	**Description**
atm no	int	20	Primary key	ATM card no.
accno	int	20	Not Null	**Account no of ATM card holders.**
pinno	int	10	Not Null	PIN no of ATM card holder.
acctype	varchar	50	Not Null	**Account type of ATM card holder.**
name	varchar	100	Not Null	Name of the ATM card holder.
balance	float		Not Null	Available balance of ATM card holder.
expiry date	int		Not Null	ATM card expiry date.

• Table Name: -

transaction It stores the information about each transaction created by ATM card holder.

Field name	**Data type**	**Size**	**Constraint**	**Description**
trid	int	10	Primary key	Transaction no
atm no	int	20	Foreign key	ATM card no.
accno	int	20	Foreign key	Account no of ATM card holder.
deposit amt	float		Not Null	Deposit amount of ATM card holder.
withamt	float		Not Null	Withdrawal amount of ATM card holder.
avl balance	float		Not Null	**Available balance of ATM card holder**.
date	date		Not Null	**Transaction date at which transaction occurred**

Data Dictionary:

The data dictionary of any system is an integral component of structure analysis, since data flow diagrams by themselves do not fully describe the subject under investigation about the system. A data dictionary is a catalog – a repository – of the elements in the system. These elements center on data and the way they are structured to meet user requirements and Money Exchange System needs. This step of creating a data dictionary is simultaneous with the process of making data flow diagram(s). Here all the data fields in their respective tables are allotted so as to access these data in the system. The data tables are created in a back-end tool like Microsoft Access, Mysql etc…. Here in the ATM System we are created database and tables using Mysql as it is the back-end tool used in the system.

The data dictionary consists of different major elements like Data Elements, Data Store [Tables Used], Data Flow, Processes and other External entities used in the system. The data dictionary stores details and description of these elements.

It is developed during data flow analysis and assists the analysts involved in determining the system requirements. Analysts use data dictionary for the following important reasons:-

- To manage the details in large system.
- To communicate a common meaning for all system elements.
- To document the features of the system.

To facilitate analysis of the details in order to evaluate the characteristics and determine where system changes should be made. To locate errors and omissions in the system. The data dictionary contains different types of descriptions for the data flowing through the system:

Data Elements is the most fundamental level which is also considered as the building block for all other data in the system. It refers to all the different data used like fields, data item, etc. to make the system fully functional irrespective to the table used in the system. Here all the different type of fields used to make table are written sequentially without referring to the tables. This process helps in the process of **Normalization** of tables.

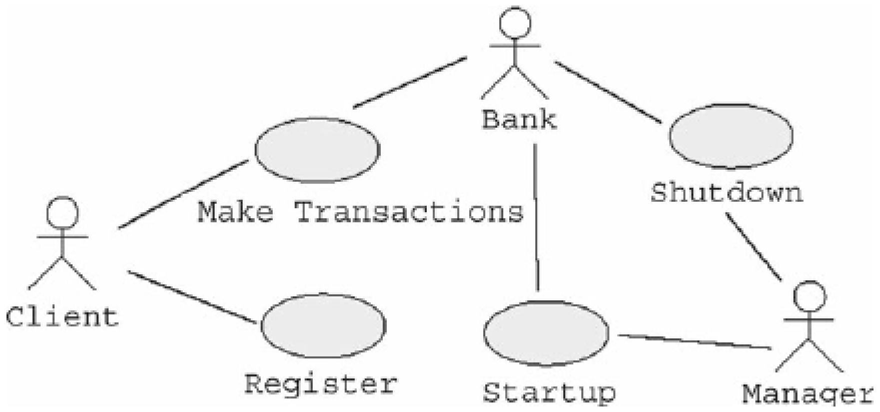
Next to Data Elements comes the **Data storage** which provides the information of where and how each data element is stored in which table and it also give information of any constraints if there. This step also gives knowledge of different data types used for different field and their size. All the normalized tables are showed in data storage.

Data Flow stage shows the flow of data in the system. This step is can be already seen in the data flow diagrams above in this document. This step refers to all the data flow paths were transactions are done in the computerized system.

The data flow step also includes different processes used in the system and it is followed by **External Entities** used in the system.

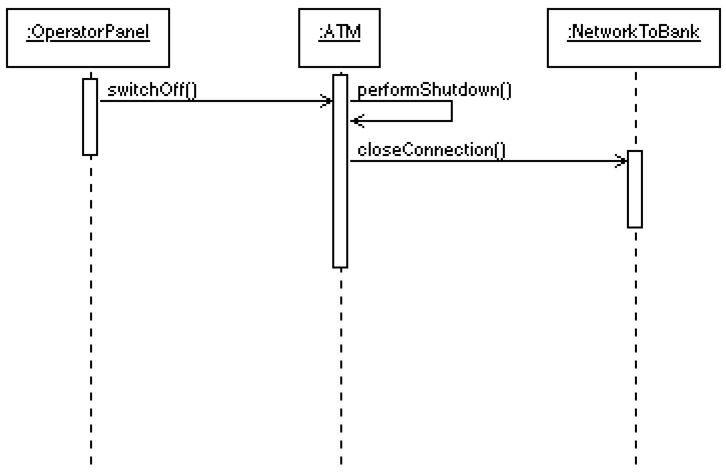
Use-case Diagrams:-

System Startup:-

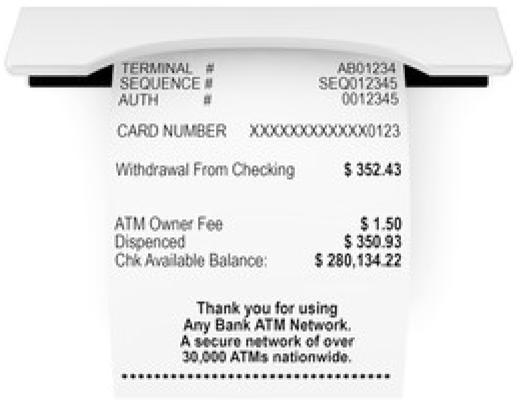


System shutdown:-

System Shutdown Sequence Diagram



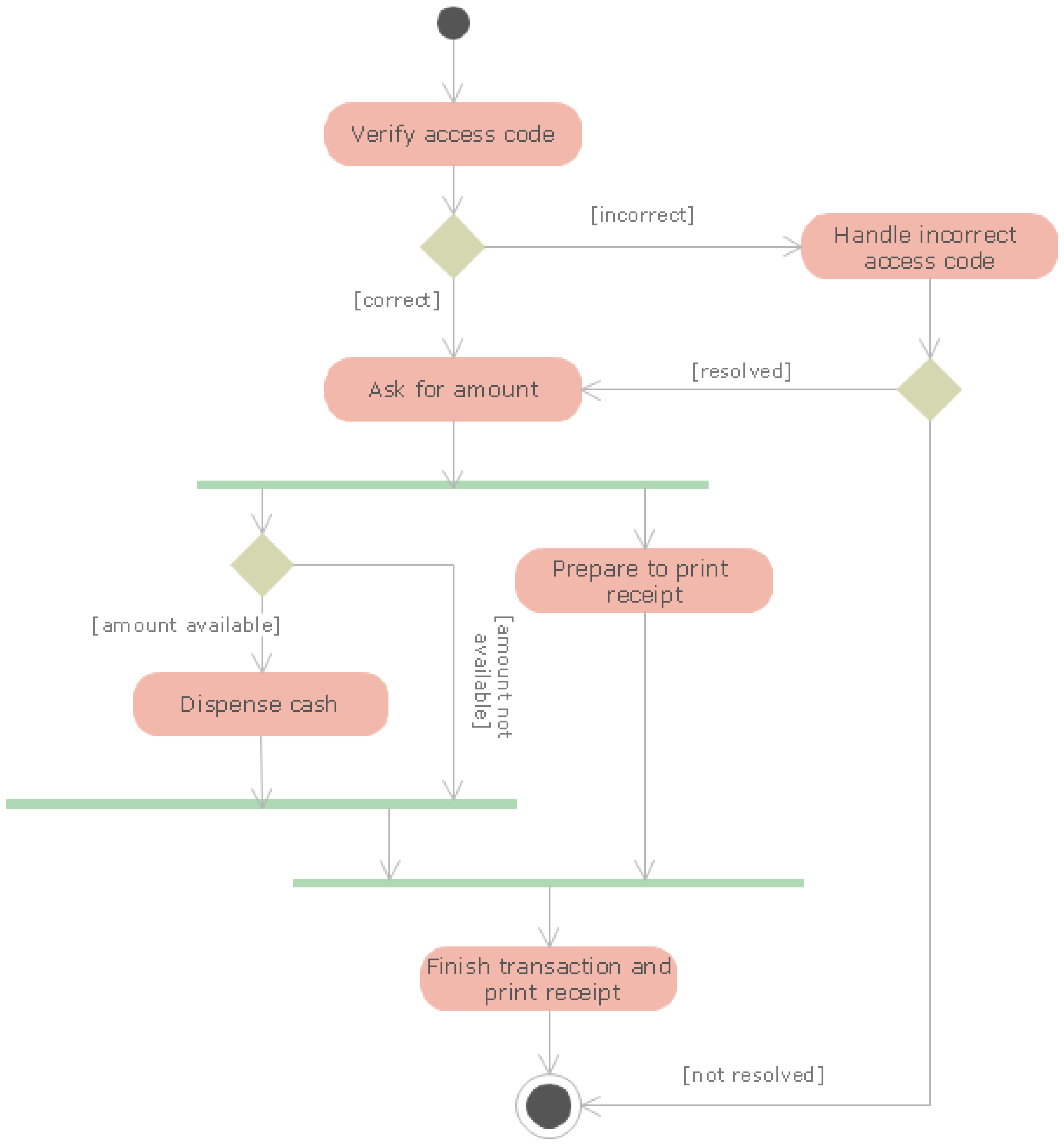
Restock cash & receipt paper:-



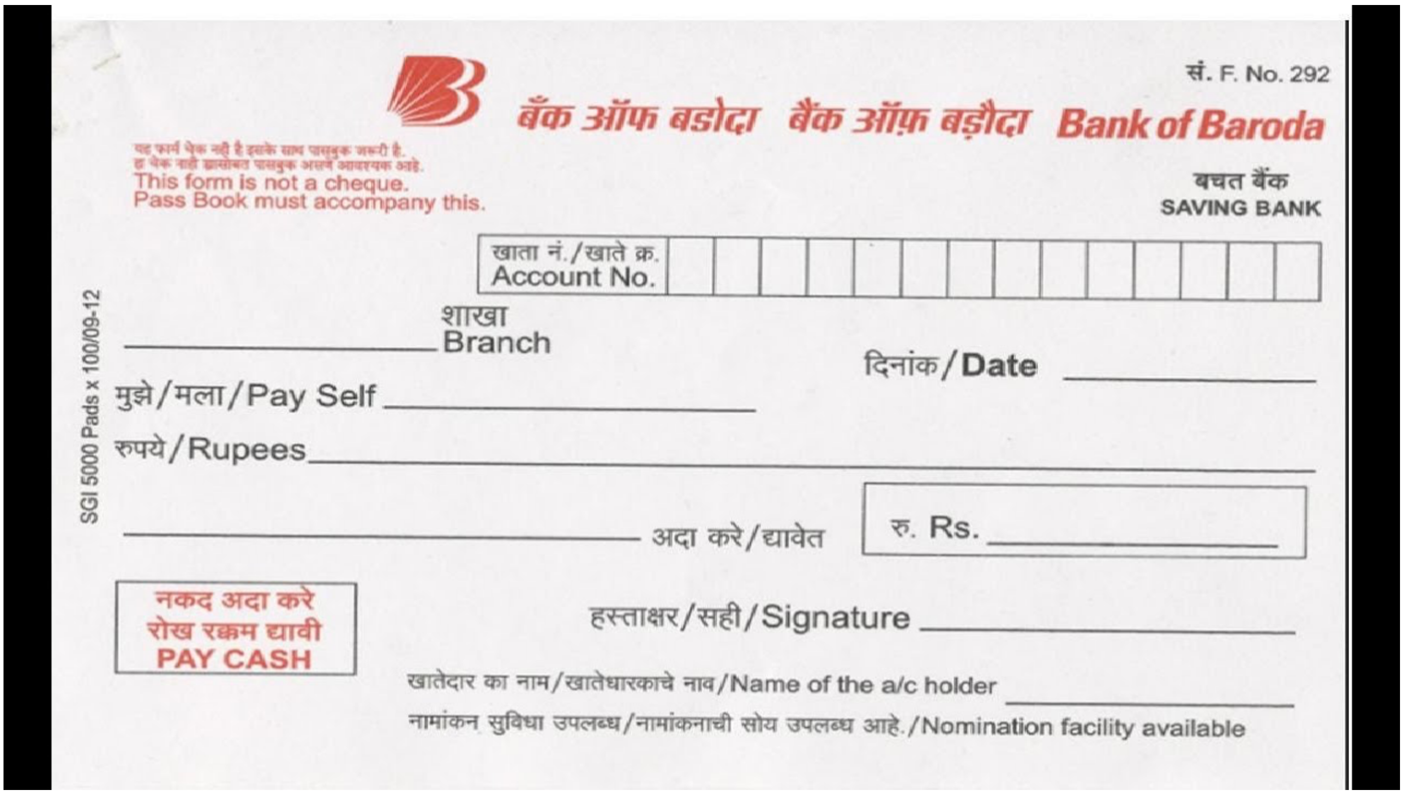
```
TERMINAL  #                 AB01234
SEQUENCE #                SEQ012345
AUTH      #                 0012345

CARD NUMBER       XXXXXXXXXXXX0123

Withdrawal From Checking    $ 352.43

ATM Owner Fee                 $ 1.50
Dispenced                   $ 350.93
Chk Available Balance:  $ 280,134.22

        Thank you for using
      Any Bank ATM Network.
      A secure network of over
       30,000 ATMs nationwide.
**********************************
```

Authenticate card ATM holder:-

Collect cash :-



Cash withdraw:-



सं. F. No. 292

बैंक ऑफ बडोदा बैंक ऑफ़ बड़ौदा Bank of Baroda

यह फार्म चेक नहीं है इसके साथ पासबुक जरूरी है.
हा चेक नाही ह्यासोबत पासबुक असणे आवश्यक आहे.
This form is not a cheque.
Pass Book must accompany this.

बचत बैंक
SAVING BANK

खाता नं./खाते क्र.
Account No.

SGI 5000 Pads x 100/09-12

________ शाखा
Branch

दिनांक/Date ________

मुझे/मला/Pay Self ________

रुपये/Rupees ________

________ अदा करे/द्यावेत

रु. Rs. ________

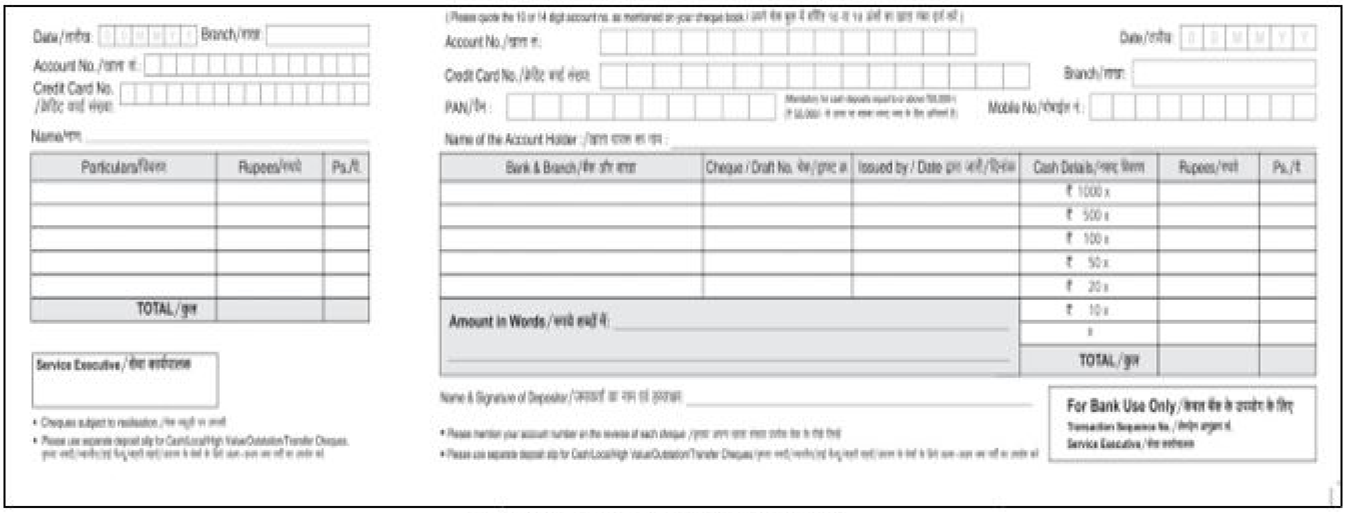
नकद अदा करे
रोख रक्कम द्यावी
PAY CASH

हस्ताक्षर/सही/Signature ________

खातेदार का नाम/खातेधारकाचे नाव/Name of the a/c holder ________

नामांकन सुविधा उपलब्ध/नामांकनाची सोय उपलब्ध आहे./Nomination facility available

Cash deposit:-



Date/तारीख: D D M M Y Y Branch/शाखा:

Account No./खाता सं.:

Credit Card No. /क्रेडिट कार्ड संख्या:

Name/नाम:

Particulars/विवरण	Rupees/रुपये	Ps./पै.
TOTAL/कुल		

Service Executive/सेवा कार्यपालक

- Cheques subject to realisation./[illegible]
- Please use separate deposit slip for Cash/Local/High Value/Outstation/Transfer Cheques. [illegible]

(Please quote the 10 or 14 digit account no. as mentioned on your cheque book./[illegible])

Account No./खाता सं.: Date/तारीख: D D M M Y Y

Credit Card No./क्रेडिट कार्ड संख्या: Branch/शाखा:

PAN/पैन: [illegible] Mobile No./मोबाइल नं.:

Name of the Account Holder :/खाता धारक का नाम :

Bank & Branch/बैंक और शाखा	Cheque / Draft No. चेक/ड्राफ्ट सं.	Issued by / Date द्वारा जारी/दिनांक	Cash Details/नकद विवरण	Rupees/रुपये	Ps./पै.
			₹ 1000 x		
			₹ 500 x		
			₹ 100 x		
			₹ 50 x		
			₹ 20 x		
Amount in Words/शब्दों में:			₹ 10 x		
			₹		
			TOTAL/कुल		

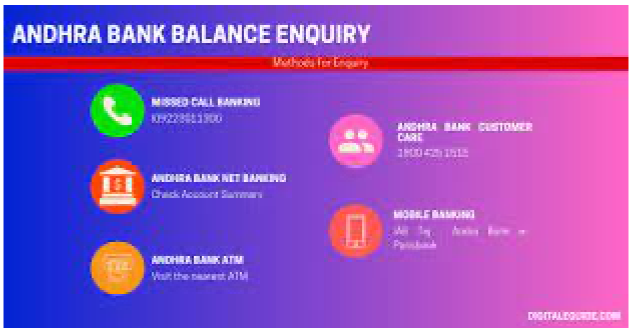
Name & Signature of Depositor/जमाकर्ता का नाम एवं हस्ताक्षर ________

- Please mention your account number on the reverse of each cheque./[illegible]
- Please use separate deposit slip for Cash/Local/High Value/Outstation/Transfer Cheques/[illegible]

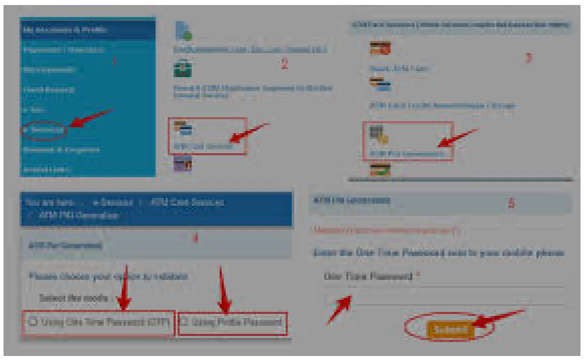
For Bank Use Only/केवल बैंक के उपयोग के लिए
Transaction Sequence No./[illegible]
Service Executive/[illegible]

Format of Cash Deposit Slip

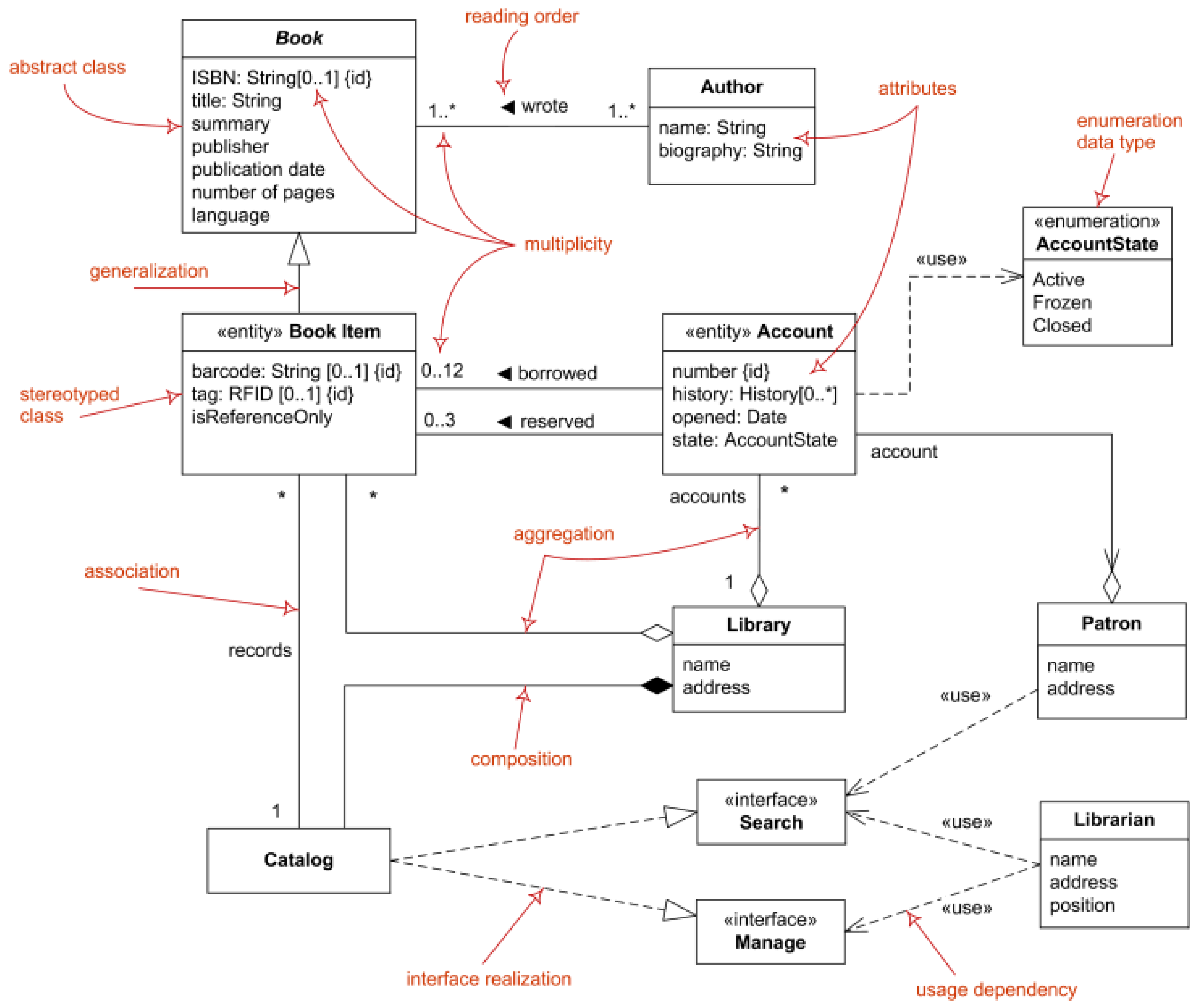
Balance Enquiry :-



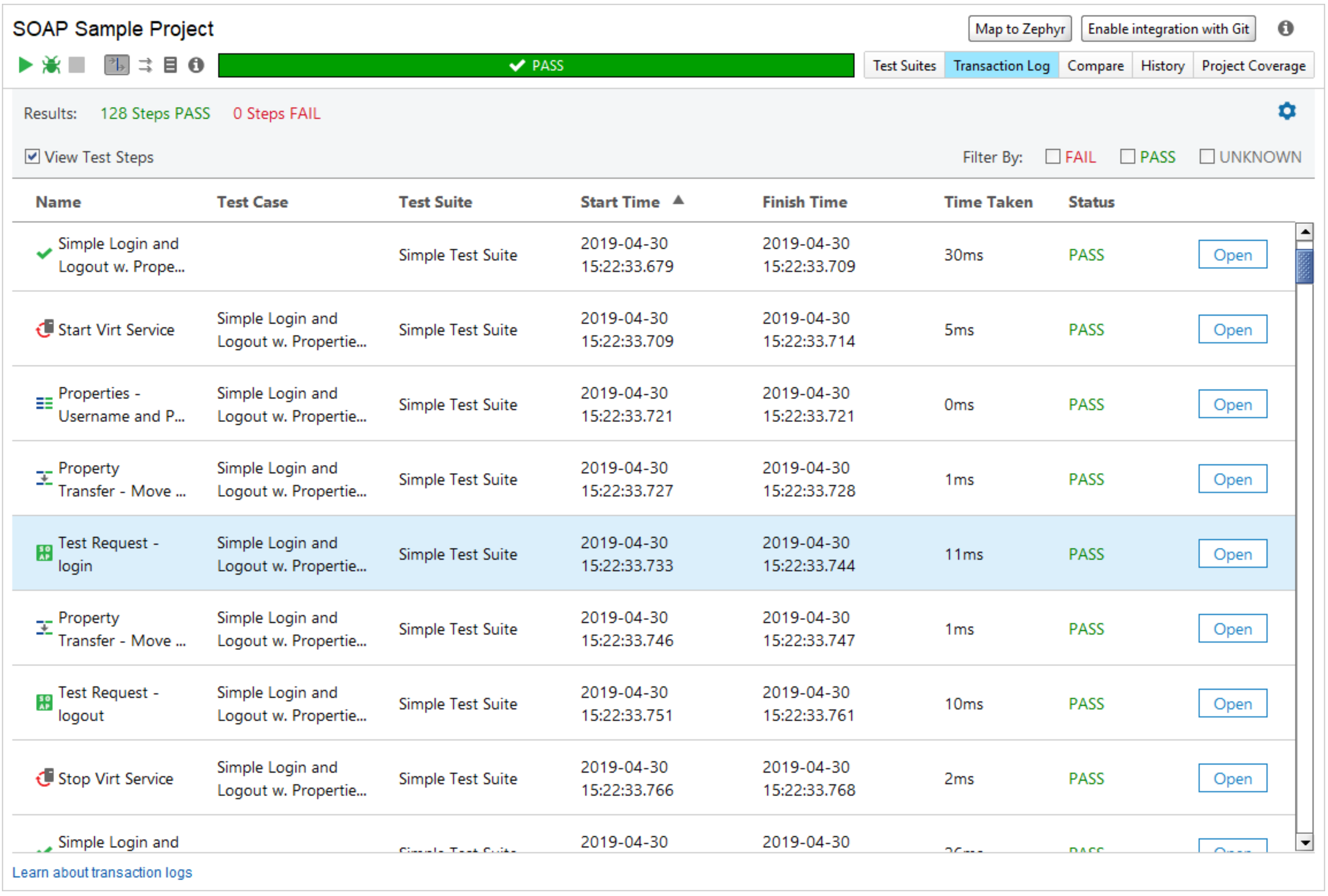
Change PIN no of ATM :-



CLASS DIAGRAM:-



Transaction:-


SOAP Sample Project

Map to Zephyr | Enable integration with Git

✓ PASS

Test Suites | Transaction Log | Compare | History | Project Coverage

Results: 128 Steps PASS 0 Steps FAIL

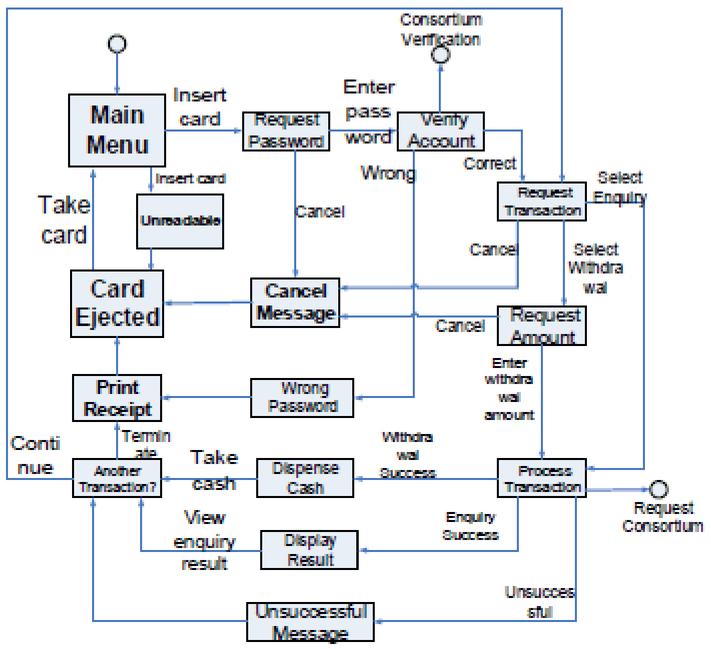
☑ View Test Steps

Filter By: ☐ FAIL ☐ PASS ☐ UNKNOWN

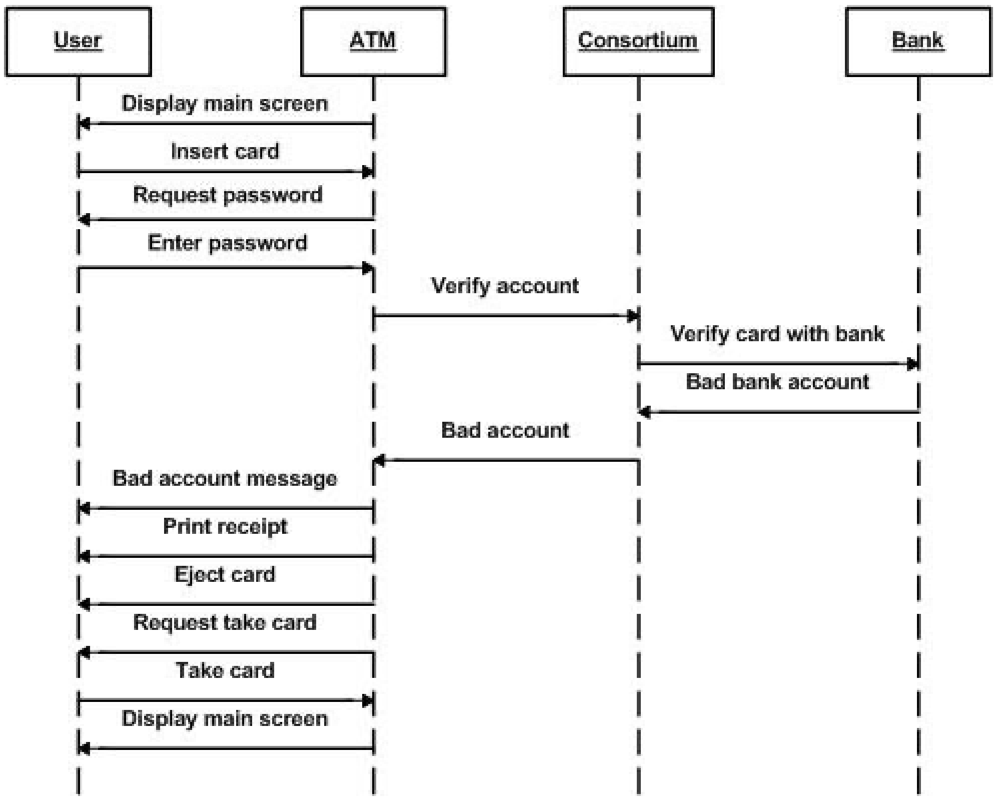
Name	Test Case	Test Suite	Start Time	Finish Time	Time Taken	Status	
✓ Simple Login and Logout w. Prope...		Simple Test Suite	2019-04-30 15:22:33.679	2019-04-30 15:22:33.709	30ms	PASS	Open
Start Virt Service	Simple Login and Logout w. Propertie...	Simple Test Suite	2019-04-30 15:22:33.709	2019-04-30 15:22:33.714	5ms	PASS	Open
Properties - Username and P...	Simple Login and Logout w. Propertie...	Simple Test Suite	2019-04-30 15:22:33.721	2019-04-30 15:22:33.721	0ms	PASS	Open
Property Transfer - Move ...	Simple Login and Logout w. Propertie...	Simple Test Suite	2019-04-30 15:22:33.727	2019-04-30 15:22:33.728	1ms	PASS	Open
Test Request - login	Simple Login and Logout w. Propertie...	Simple Test Suite	2019-04-30 15:22:33.733	2019-04-30 15:22:33.744	11ms	PASS	Open
Property Transfer - Move ...	Simple Login and Logout w. Propertie...	Simple Test Suite	2019-04-30 15:22:33.746	2019-04-30 15:22:33.747	1ms	PASS	Open
Test Request - logout	Simple Login and Logout w. Propertie...	Simple Test Suite	2019-04-30 15:22:33.751	2019-04-30 15:22:33.761	10ms	PASS	Open
Stop Virt Service	Simple Login and Logout w. Propertie...	Simple Test Suite	2019-04-30 15:22:33.766	2019-04-30 15:22:33.768	2ms	PASS	Open
Simple Login and			2019-04-30	2019-04-30			

Learn about transaction logs

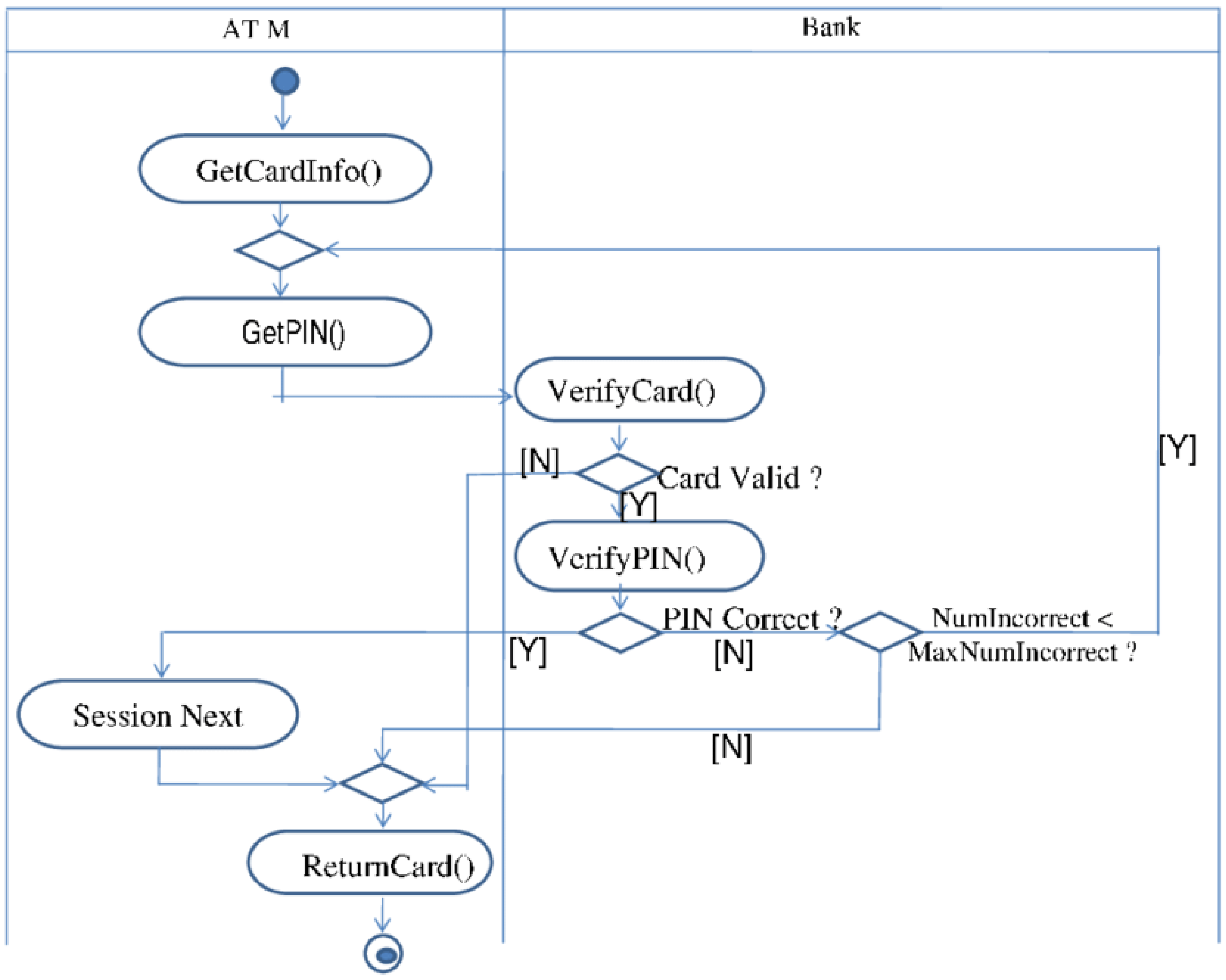
ATM SYSTEM:-



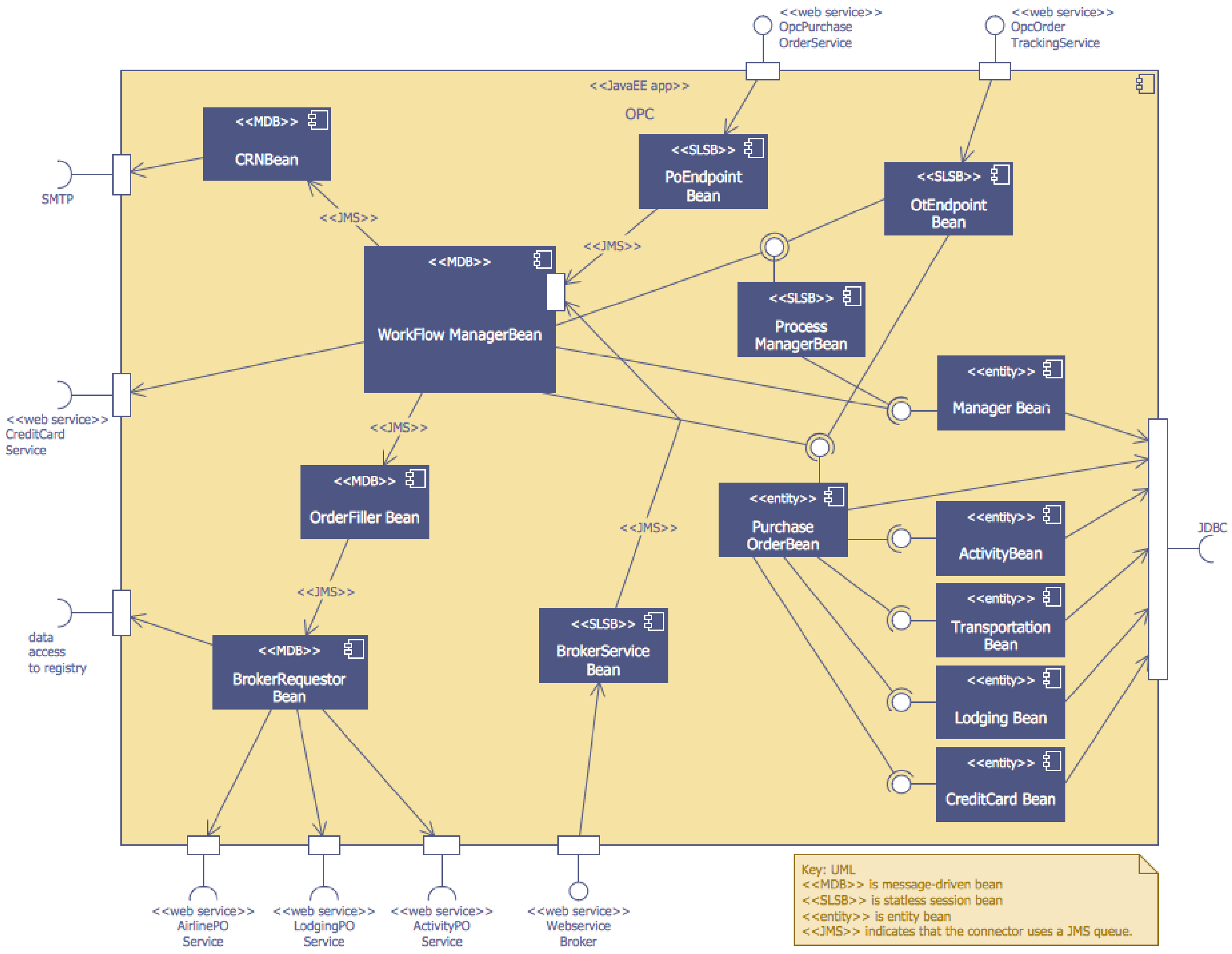
Sequence Diagram :



Activity Diagram:-

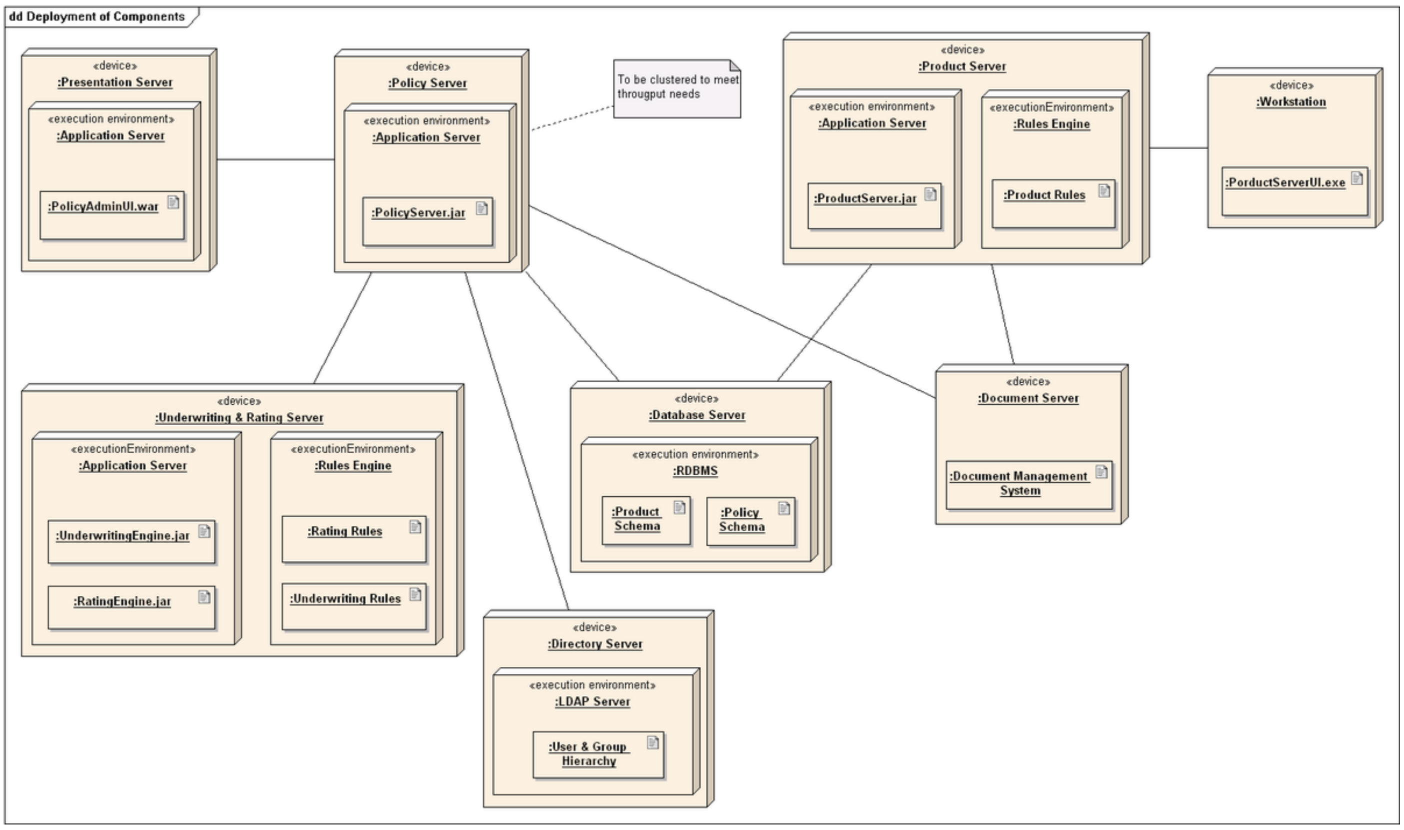

AT M
Bank
GetCardInfo()
GetPIN()
VerifyCard()
[N]
Card Valid ?
[Y]
VerifyPIN()
PIN Correct ?
[Y]
[N]
NumIncorrect <
MaxNumIncorrect ?
[Y]
[N]
Session Next
ReturnCard()

Component Diagram:-


<<web service>> OpcPurchase OrderService
<<web service>> OpcOrder TrackingService
<<JavaEE app>>
OPC
<<MDB>> CRNBean
<<SLSB>> PoEndpoint Bean
<<SLSB>> OtEndpoint Bean
SMTP
<<JMS>>
<<JMS>>
<<MDB>> WorkFlow ManagerBean
<<SLSB>> Process ManagerBean
<<entity>> Manager Bean
<<web service>> CreditCard Service
<<JMS>>
<<MDB>> OrderFiller Bean
<<entity>> Purchase OrderBean
<<JMS>>
<<entity>> ActivityBean
JDBC
<<JMS>>
<<entity>> Transportation Bean
data access to registry
<<MDB>> BrokerRequestor Bean
<<SLSB>> BrokerService Bean
<<entity>> Lodging Bean
<<entity>> CreditCard Bean
<<web service>> AirlinePO Service
<<web service>> LodgingPO Service
<<web service>> ActivityPO Service
<<web service>> Webservice Broker

Key: UML
<<MDB>> is message-driven bean
<<SLSB>> is stateless session bean
<<entity>> is entity bean
<<JMS>> indicates that the connector uses a JMS queue.

Deployment Diagram:-

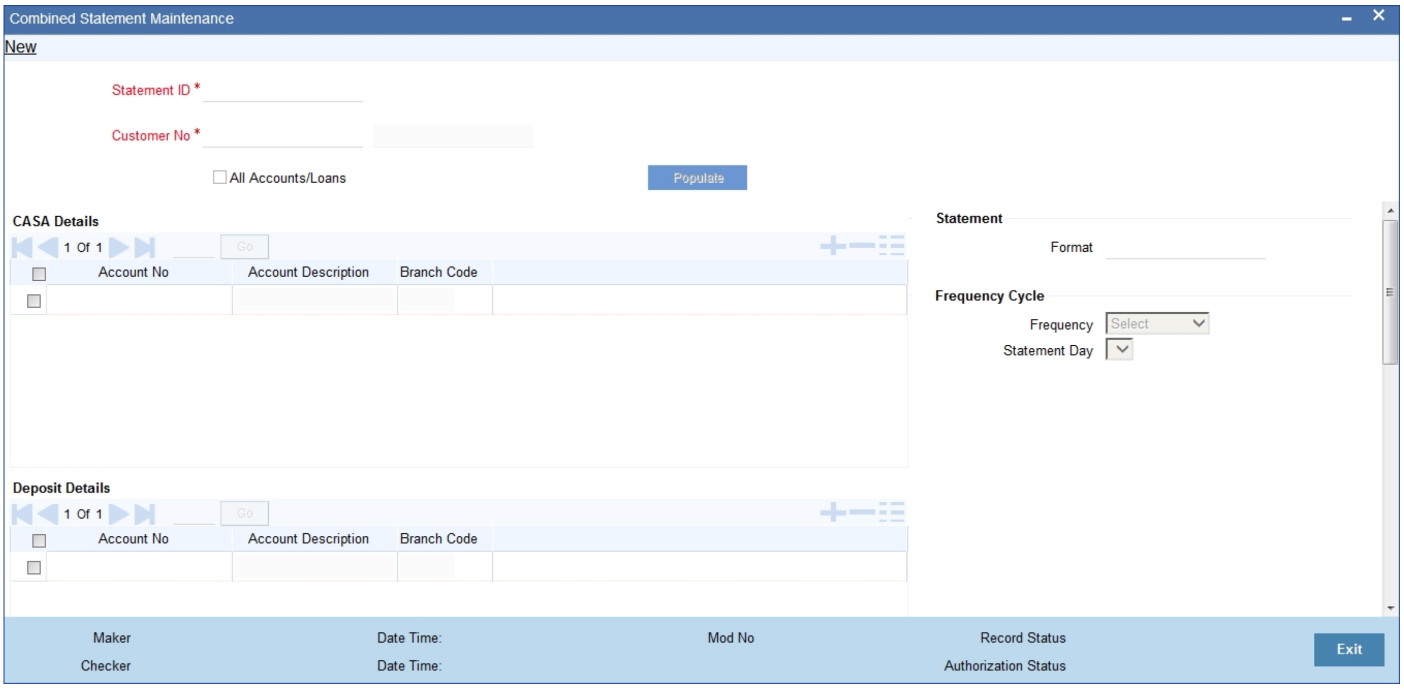

dd Deployment of Components
«device» :Presentation Server
«execution environment» :Application Server
:PolicyAdminUI.war
«device» :Policy Server
«execution environment» :Application Server
:PolicyServer.jar
To be clustered to meet througput needs
«device» :Product Server
«execution environment» :Application Server
:ProductServer.jar
«executionEnvironment» :Rules Engine
:Product Rules
«device» :Workstation
:PorductServerUI.exe
«device» :Underwriting & Rating Server
«executionEnvironment» :Application Server
:UnderwritingEngine.jar
:RatingEngine.jar
«executionEnvironment» :Rules Engine
:Rating Rules
:Underwriting Rules
«device» :Database Server
«execution environment» :RDBMS
:Product Schema
:Policy Schema
«device» :Document Server
:Document Management System
«device» :Directory Server
«execution environment» :LDAP Server
:User & Group Hierarchy

Use interface design: - Input Screens using sample data:

- Welcome screen:
- ATM card no screen: on success then below screen.
- ATM card no screen: if enter wrong ATM no or PIN no.
- ATM card no screen: if ATM card is out expiry date
- ATM card no screen: each screen display out of 20sec then below message.
- Account type screen: if account type is correct then go to transaction menu & wrong then appear below screen & go to welcome screen.
- Transaction menu screen:

- Cash withdrawal screen: if cash withdrawal is success then appear below screen, if we click on YES then go to balance enquiry screen, if NO then show available balance.
- Cash withdrawal screen: if we enter amount >10000 then below appear screen.
- Cash withdrawal screen: if per day withdrawal amount >25000 then below appear screen if your account is saving & account is current then amount>50000.
- Cash deposit screen: if cash deposit is success then appear below screen, if we click on YES then go to balance enquiry screen, if NO then show available balance.
- Balance Enquiry screen:
- PIN change screen: if we enter old pin no correct and new & confirm pin no same then appear below screen.
- PIN change screen: if we enter old pin no wrong and new & confirm pin no same then appear below screen.
- PIN change screen: if we enter old pin no correct and new & confirm pin no different then appear below screen.
- Loan Information:
- Help screen:

<u>Reports: 1) Mini statement screen:-</u>


Combined Statement Maintenance

New

Statement ID *

Customer No *

☐ All Accounts/Loans

Populate

CASA Details

☐	Account No	Account Description	Branch Code
☐			

Deposit Details

☐	Account No	Account Description	Branch Code
☐			

Statement

Format

Frequency Cycle

Frequency: Select

Statement Day

Maker	Date Time:	Mod No	Record Status
Checker	Date Time:		Authorization Status

Exit

Testing & Implementation plan: -

A process of executing a program with the goal of finding errors is Software Testing. A Software Testing Strategy helps to convert test case designs into a well-planned execution steps that will result in the construction of successful software. Software testing is a destructive process of trying to find the errors. The main purpose of testing can be quality assurance, reliability estimation, validation or verification.

Principle of Testing:

- All tests should be traceable as per the customer requirement.
- Resource planning should be in advanced.
- Test cases should not leave any fatal error.
- Test case should handle all critical functions.

Testing is the set of activities that can be planned in advance and conducted systematically. To define these activities templates are provided by different testing strategies.

Unit Testing:

At vertex of spiral, testing begins with unit testing. It aims at testing each component or unit of software to check its functionality, independently. Ensures that it works properly as a unit. Typical units are
Interface: tested to check proper flow of information into and out of the program unit under test.

Local data structures:

tested to check integrity of data during execution. Boundary conditions: tested to ensure unit operates properly at boundaries to limit processing.
Independent paths: tested to ensure all statements in the unit are executed at least once.

Error handling paths:

tested to check whether error messages are user friendly and corresponds to error encountered, whether they reroute or terminate process when error occurred.

Common errors found during unit testing are: incorrect initialization, precision inaccuracy, mixed mode operation, incorrect arithmetic precedence etc.

Integration testing:

Further progressing the testing process, these units must be assembled or integrated to form complete software package. So integration testing focuses the problems of verification and construction.

Validation testing:

Taking one more outward turn along spiral, comes validation testing. It consists of higher order tests using validation criteria defined during requirement analysis phase. This test assures that software meets all functional, behavioural and performance requirements.

Performance Testing:

It concentrate on the transaction response time, throughput etc. It is designed to test the run-time performance of software within the context of an integral system. Performance testing is conducted throughout all steps of testing process.

User Acceptance Testing:

The major concern while developing any product is that the product must satisfy the user. Any acceptance testing validates & verifies whether the product is user acceptable or not. This test confirms the reliability of the product.

CHAPTER 4

CODING

```
#!/usr/bin/env python
from django.core.management import execute_manager
import imp
try:
    imp.find_module('settings') # Assumed to be in the same directory.
except ImportError:
    import sys
    sys.stderr.write("Error: Can't find the file 'settings.py' in the directory
containing %r. It appears you've customized things.\nYou'll have to run
django-admin.py, passing it your settings module.\n" % __file__)
    sys.exit(1)

import settings

if __name__ == "__main__":
    execute_manager(settings)
# Django settings for project project.

DEBUG = True
TEMPLATE_DEBUG = DEBUG

ADMINS = (
    # ('Your Name', 'your_email@example.com'),
)

MANAGERS = ADMINS

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3', # Add 'postgresql_psycopg2',
'postgresql', 'mysql', 'sqlite3' or 'oracle'.
        'NAME': 'atm.db',                      # Or path to database file if using sqlite3.
        'USER': '',                      # Not used with sqlite3.
        'PASSWORD': '',                  # Not used with sqlite3.
        'HOST': '',                  # Set to empty string for localhost. Not used with
sqlite3.
        'PORT': '',                  # Set to empty string for default. Not used with
sqlite3.
    }
}

# Local time zone for this installation. Choices can be found here:
# http://en.wikipedia.org/wiki/List_of_tz_zones_by_name
# although not all choices may be available on all operating systems.
# On Unix systems, a value of None will cause Django to use the same
# timezone as the operating system.
# If running in a Windows environment this must be set to the same as your
# system time zone.
TIME_ZONE = 'America/Chicago'

# Language code for this installation. All choices can be found here:
# http://www.i18nguy.com/unicode/language-identifiers.html
LANGUAGE_CODE = 'en-us'

SITE_ID = 1

# If you set this to False, Django will make some optimizations so as not
# to load the internationalization machinery.
USE_I18N = True

# If you set this to False, Django will not format dates, numbers and
# calendars according to the current locale
USE_L10N = True

# Absolute filesystem path to the directory that will hold user-uploaded files.
# Example: "/home/media/media.lawrence.com/media/"
MEDIA_ROOT = ''

# URL that handles the media served from MEDIA_ROOT. Make sure to use a
# trailing slash.
# Examples: "http://media.lawrence.com/media/", "http://example.com/media/"
MEDIA_URL = ''

# Absolute path to the directory static files should be collected to.
# Don't put anything in this directory yourself; store your static files
# in apps' "static/" subdirectories and in STATICFILES_DIRS.
# Example: "/home/media/media.lawrence.com/static/"
STATIC_ROOT = ''

# URL prefix for static files.
# Example: "http://media.lawrence.com/static/"
STATIC_URL = '/static/'

# URL prefix for admin static files -- CSS, JavaScript and images.
# Make sure to use a trailing slash.
# Examples: "http://foo.com/static/admin/", "/static/admin/".
ADMIN_MEDIA_PREFIX = '/static/admin/'

# Additional locations of static files
STATICFILES_DIRS = (
    # Put strings here, like "/home/html/static" or "C:/www/django/static".
    # Always use forward slashes, even on Windows.
    # Don't forget to use absolute paths, not relative paths.
)

# List of finder classes that know how to find static files in
# various locations.
STATICFILES_FINDERS = (
    'django.contrib.staticfiles.finders.FileSystemFinder',
    'django.contrib.staticfiles.finders.AppDirectoriesFinder',
#    'django.contrib.staticfiles.finders.DefaultStorageFinder',
)

# Make this unique, and don't share it with anybody.
```

```
SECRET_KEY = 'p@jbyw1wc2rhp9gogcj(=kcv7l3dzjw^+p!bm2!1uei6_++r+5'

# List of callables that know how to import templates from various sources.
TEMPLATE_LOADERS = (
    'django.template.loaders.filesystem.Loader',
    'django.template.loaders.app_directories.Loader',
#     'django.template.loaders.eggs.Loader',
)

MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.middleware.csrf.CsrfResponseMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
)

ROOT_URLCONF = 'project.urls'

TEMPLATE_DIRS = ("templates",
    # Always use forward slashes, even on Windows.
    # Don't forget to use absolute paths, not relative paths.
)

INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'version1',
    # Uncomment the next line to enable the admin:
    'django.contrib.admin',
    'tastypie',
    # Uncomment the next line to enable admin documentation:
    # 'django.contrib.admindocs',
)

# A sample logging configuration. The only tangible logging
# performed by this configuration is to send an email to
# the site admins on every HTTP 500 error.
# See http://docs.djangoproject.com/en/dev/topics/logging for
# more details on how to customize your logging configuration.
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'mail_admins': {
            'level': 'ERROR',
            'class': 'django.utils.log.AdminEmailHandler'
        }
    },
    'loggers': {
        'django.request': {
            'handlers': ['mail_admins'],
            'level': 'ERROR',
            'propagate': True,
        },
    }
}
rom django.conf.urls.defaults import patterns, include, url
from django.contrib.staticfiles.urls import staticfiles_urlpatterns
from version1.api import *
from tastypie.api import Api
from django.contrib import admin

admin.autodiscover()
```

```
v1_api = Api(api_name='v1')
v1_api.register(CashWithdrawalResource())
v1_api.register(CashTransferResource())
v1_api.register(ServicesResource())
v1_api.register(ATMCardResource())

urlpatterns = patterns('',
    url(r'^user/$', 'version1.views.main'),
    url(r'^user/card/$', 'version1.views.index'),
    url(r'^user/validatepin/$', 'version1.views.validatepin'),
    url(r'^user/validatepasscode/$', 'version1.views.validatepasscode'),
    url(r'^user/options/$', 'version1.views.options'),
    url(r'^user/history/$', 'version1.views.history'),
    url(r'^user/balanceenquiry/$', 'version1.views.balanceenquiry'),
    url(r'^user/cashwithdrawal/$', 'version1.views.cashwithdrawal'),
    url(r'^user/cashtransfer/$', 'version1.views.cashtransfer'),
    url(r'^user/pinchange/$', 'version1.views.pinchange'),
    url(r'^user/phonechange/$', 'version1.views.phonechange'),
    url(r'^user/fastcash/$', 'version1.views.fastcash'),
    url(r'^user/services/$', 'version1.views.services'),
    url(r'^user/exit/$', 'version1.views.exit'),
    url(r'^admin/', include(admin.site.urls)),
    url(r'^api/', include(v1_api.urls)),
    url(r'^BankServices/rest/biller/$', 'version1.views.services_mock_listBiller'),
    url(r'^admin_user/$', 'version1.views.admin_index'),
    url(r'^admin_user/logout/$', 'version1.views.admin_logout'),
    url(r'^admin_user/verify_user/$', 'version1.views.admin_verify_user'),
    url(r'^admin_user/main_page/$', 'version1.views.admin_main_page'),
    url(r'^admin_user/add_new_card/$', 'version1.views.admin_add_card'),
    url(r'^admin_user/add_atm_operation/$',
'version1.views.admin_add_card_operation'),
    url(r'^admin_user/ATM_status/$', 'version1.views.admin_atm_status'),
    url(r'^admin_user/update_refill/$', 'version1.views.admin_update_refill'),
    url(r'^admin_user/update_card_details/$',
'version1.views.admin_update_card_details'),
   url(r'^admin_user/update_card_details/validate_card/$',
'version1.views.admin_card_validation'),
   url(r'^admin_user/update_card_details/main_page/$',
'version1.views.admin_update_card_main_page'),
   url(r'^admin_user/update_card_details/block_card/$',
'version1.views.admin_block_card'),
   url(r'^admin_user/update_card_details/block_card_operation/$',
'version1.views.admin_block_card_operation'),
   url(r'^admin_user/update_card_details/activate_card/$',
'version1.views.admin_activate_card'),
   url(r'^admin_user/update_card_details/activate_card_operation/$',
'version1.views.admin_activate_card_operation'),
   url(r'^admin_user/update_card_details/reset_pincode/$',
'version1.views.admin_reset_pincode'),
   url(r'^admin_user/update_card_details/reset_pincode_operation/$',
'version1.views.admin_reset_pincode_operation'),
   url(r'^admin_user/update_card_details/reset_phone/$',
'version1.views.admin_reset_phone'),
   url(r'^admin_user/update_card_details/reset_phone_operation/$',
'version1.views.admin_reset_phone_operation'),
   url(r'^admin_user/update_card_details/view_history/$',
'version1.views.admin_view_history'),
   url(r'^admin_user/update_card_details/update_date/$',
'version1.views.admin_update_date'),
   url(r'^admin_user/update_card_details/update_date_operation/$',
'version1.views.admin_update_date_operation'),
)

urlpatterns += staticfiles_urlpatterns()
rom version1.models import Admin
from version1.models import Machine
from version1.models import MachineRefill
from version1.models import Account_Ext
from version1.models import ATM_Card
from version1.models import Balance_Enquiry
from version1.models import Cash_Transfer
from version1.models import Cash_Withdrawl
from version1.models import Pin_change
from version1.models import Phone_change
from django.contrib import admin

admin.site.register(Admin)
admin.site.register(Machine)
admin.site.register(MachineRefill)
admin.site.register(Account_Ext)
admin.site.register(ATM_Card)
admin.site.register(Balance_Enquiry)
admin.site.register(Cash_Transfer)
admin.site.register(Cash_Withdrawl)
admin.site.register(Pin_change)
admin.site.register(Phone_change)
# version1/api.py
from tastypie.resources import ModelResource
from version1.models import Cash_Withdrawl, Cash_Transfer, Services,
ATM_Card

class CashWithdrawalResource(ModelResource):
        class Meta:
                queryset = Cash_Withdrawl.objects.filter(rescode=2)
                resource_name = 'cashwithdrawal'

class CashTransferResource(ModelResource):
        class Meta:
                queryset = Cash_Transfer.objects.filter(rescode=3)
                resource_name = 'cashtransfer'

class ServicesResource(ModelResource):
        class Meta:
                queryset = Cash_Transfer.objects.filter(rescode=101)
                resource_name = 'service'
```

```
class ATMCardResource(ModelResource):
        class Meta:
                queryset = ATM_Card.objects.all()
                resource_name = 'atmcard'
                excludes = ['pin', 'expiry_date']
from django.db import models
from decimal import *

## Admin model stores the information of adiministrator.
## It has 2 fields
## Admin_id -->char(100)
## password -->char(100)
class Admin(models.Model):
        Admin_id = models.CharField("ADMIN
ID",max_length=100,primary_key=True)
        Password = models.CharField("PASSWORD",max_length=100)
        def __unicode__(self):
                return str(self.Admin_id)

## Machine model stores the information of ATM_Machine
## It has 6 fields
## location ---> Char(200)
## minimum_atm_balance ---> Decimal
## current_balance ----> Decimal
## last_refill_date ----> DateTimeField
## next_maintainence_date ----> DateTimeField
class Machine(models.Model):
        machine_id = models.IntegerField("MACHINE ID",primary_key=True)
        location = models.CharField("LOCATION",max_length=200)
        minimum_atm_balance = models.DecimalField("MINIMUM ATM
BALANCE",decimal_places=2,max_digits=10)
        current_balance =
models.DecimalField("BALANCE",decimal_places=2,max_digits=10)
        last_refill_date = models.DateTimeField('LAST REFILL DATE')
        next_maintainence_date = models.DateTimeField('NEXT
MAINTAINENCE DATE')
        def __unicode__(self):
                return str(self.machine_id)

        def save(self, *args, **kwargs):
                if(self.machine_id<=0):
                        raise Exception, "MACHINE ID SHOULD BE NON ZERO
POSITIVE"
                else:
                        if(self.minimum_atm_balance<0):
                                raise Exception, "MINIMUM ATM BALANCE SHOULD
BE POSITIVE"
                        else:
                                if(self.current_balance<0):
                                        raise Exception, "BALANCE SHOULD BE
POSITIVE"
                                else:
                                        super(Machine, self).save()

## MachineRefill model stores the information of refill for ATM machine.
## It has 5 fields
## refill_id -->Integer
## machine_id ---> ForeignKey of Machine
## refill_date ---> DateTimeField
## previous_balance ---> DecimalField
## amount_refilled ----> Decimal
class MachineRefill(models.Model):
        refill_id = models.IntegerField("REFILL ID",primary_key=True)
        machine_id = models.ForeignKey(Machine)
        refill_date = models.DateTimeField('REFILL DATE')
        previous_balance = models. DecimalField("PREVIOUS ATM
BALANCE",decimal_places=2,max_digits=10)
        amount_refilled = models.DecimalField("AMOUNT
REFILLED",decimal_places=2,max_digits=10)
```

```
    def __unicode__(self):
        return str(self.refill_id)
    def save(self, *args, **kwargs):
        if(self.refill_id<=0):
            raise Exception, "REFILL ID SHOULD BE NON ZERO
POSITIVE"
        else:
            if(self.previous_balance<0):
                raise Exception, "PREVIOUS ATM BALANCE SHOULD
BE POSITIVE"
            else:
                if(self.amount_refilled<0):
                    raise Exception, "AMOUNT REFILLED SHOULD
BE POSITIVE"
                else:
                    super(MachineRefill, self).save()

## Account_Ext model stores the information of account.
## It has 5 fields
## acc_num = BigInteger
## atmcard_num ---> BigInteger
## name --> Char
## phone_num --> BigInteger
## balance --> Decimal
class Account_Ext(models.Model):
    acc_num = models.BigIntegerField("Account Number",primary_key=True)
    name = models.CharField("NAME",max_length=100)
    phone_num = models.BigIntegerField("Phone NUMBER")
    balance =
models.DecimalField("Balance",decimal_places=2,max_digits=10)
    def __unicode__(self):
        return str(self.acc_num)
    def save(self, *args, **kwargs):
        if(self.acc_num > 0):
            if(len(str(self.acc_num)) == 12):
```

```
                                if(self.phone_num<=0):
                                        raise Exception, "PHONE NUMBER SHOULD BE
NON ZERO POSITIVE"
                                else:
                                        if(len(str(self.phone_num)) != 10):
                                                raise Exception, "PHONE NUMBER
LENGTH SHOULD BE 10"
                                        else:
                                                if(self.balance<0):
                                                        raise Exception, "BALANCE SHOULD
BE POSITIVE"
                                                else:
                                                        super(Account_Ext, self).save()
                        else:
                                raise Exception, "ACCOUNT NUMBER LENGTH
SHOULD BE 12"
                else:
                        raise Exception, "ACCOUNT NUMBER SHOULD BE NON
ZERO POSITIVE"

## ATM_Card model stores the information of ATM cards
## It has 9 fields
## atmcard_num ---> ForeignKey of Account_Ext
## name ---> Char(100)
## pin ---> Integer
## date_of_issue ---> DateTimeField
## expiry_date ---> DateTimeField
## address --> Char(300)
## two_factor ---> Boolean
## phone_num ---> BigInteger
## card_status ---> Boolean
class ATM_Card(models.Model):
        atmcard_num = models.BigIntegerField("ATM Card Number",
primary_key=True)
        account_num = models.ForeignKey(Account_Ext)
```

```
	name = models.CharField("NAME ON CARD",max_length=100)
	pin = models.IntegerField("PIN", max_length=4)
	date_of_issue = models.DateTimeField('DATE OF ISSUE')
	expiry_date = models.DateTimeField('EXPIRY DATE')
	address = models.CharField("ADDRESS",max_length=300)
	two_factor = models.BooleanField("TWO FACTOR AUTHENTICATION
STATUS")
	phone_num = models.BigIntegerField("PHONE NUMBER FOR
AUTHENTICATION")
	card_status = models.BooleanField("CARD STATUS")
	def __unicode__(self):
		return str(self.atmcard_num)

	def save(self, *args, **kwargs):
		if Account_Ext.objects.filter(acc_num =
self.account_num_id).count() == 0:
			raise Exception, "Account does not exist"
		if(self.atmcard_num > 0):
			if(len(str(self.atmcard_num)) == 4):
				if(self.pin <= 0):
					raise Exception, "PIN NUMBER SHOULD BE
NON ZERO POSITIVE"
				else:
					if(len(str(self.pin)) != 4):
						raise Exception, "PIN NUMBER LENGTH
SHOULD BE 4"
					else:
						if(self.expiry_date<=self.date_of_issue):
							raise Exception, "EXPIRY DATE
SHOULD BE AFTER THE ISSUE DATE"
						else:
							if(self.phone_num<=0):
								raise Exception, "PHONE
NUMBER SHOULD BE NON ZERO POSITIVE"
							else:
```

```
                                        if(len(str(self.phone_num))!=10):
                                             raise Exception, "PHONE
NUMBER LENGTH SHOULD BE 10"
                                        else:
                                             super(ATM_Card, self).save()
                    else:
                         raise Exception, "ATM CARD NUMBER LENGTH
SHOULD BE 4"
               else:
                    raise Exception, "ATM CARD NO. SHOULD BE NON ZERO
POSITIVE"

## Transaction is the abstract class which has sub-classes
## It has 7 fields
## atmcard_num ---> ForeignKey of Account_Ext
## machine_id ---> ForeignKey of Machine
## tid ---> Integer
## date_time ---> DateTimeField
## status ---> Char(100)
## rescode ---> Integer
## type_trans ---> Char(100)
class Transaction(models.Model):
     atmcard_num = models.ForeignKey(ATM_Card)
     machine_id = models.ForeignKey(Machine)
     tid = models.IntegerField("TRANSACTION ID",primary_key=True)
     date_time = models.DateTimeField("DATE TIME OF TRANSACTION")
     status = models.CharField("STATUS",max_length = 100)
     rescode = models.IntegerField("RESPONSE CODE")
     type_trans = models.CharField("TRANSACTION TYPE",max_length =
100)
     class Meta:
          abstract = True

## Balance_Enquiry() is the inherited class of trasaction class
## It has 1 fields
## bal_amount ----> Decimal
class Balance_Enquiry(Transaction):
        bal_amount = models.DecimalField("BALANCE
AMOUNT",decimal_places=2,max_digits=10)
        def __unicode__(self):
                return str(self.tid)
        def save(self, *args, **kwargs):
                if(self.tid<=0):
                        raise Exception, "TRANSACTION ID SHOULD BE NON
ZERO POSITIVE"
                else:
                        if(self.rescode<0):
                                raise Exception, "RESPONSE CODE SHOULD BE
POSITIVE"
                        else:
                                if(self.bal_amount<0):
                                        raise Exception, "BALANCE AMOUNT SHOULD
BE POSITIVE"
                                else:
                                        count=0
                                        for e in Balance_Enquiry.objects.all():
                                                count=count+1
                                        for e in Cash_Withdrawl.objects.all():
                                                count=count+1
                                        for e in Cash_Transfer.objects.all():
                                                count=count+1
                                        for e in Phone_change.objects.all():
                                                count=count+1
                                        for e in Pin_change.objects.all():
                                                count=count+1
                                        self.tid = count+1
                                        super(Balance_Enquiry, self).save()

## phone_change() is the inherited class of trasaction class
## It has 2 fields
## prev_phone ---> BigInteger
## new_phone ---> BigInteger
class Phone_change(Transaction):
        prev_phone = models.BigIntegerField("PREVIOUS PHONE NO")
        new_phone = models.BigIntegerField("NEW PHONE NO")
        def __unicode__(self):
                return str(self.tid)
        def save(self, *args, **kwargs):
                if(self.tid<=0):
                        raise Exception, "TRANSACTION ID SHOULD BE NON
ZERO POSITIVE"
                else:
                        if(self.rescode<0):
                                raise Exception, "RESPONSE CODE SHOULD BE
POSITIVE"
                        else:
                                if(self.prev_phone<=0):
                                        raise Exception, "PREVIOUS PHONE NO
SHOULD BE POSITIVE"
                                else:
                                        if(len(str(self.prev_phone))!=10):
                                                raise Exception, "PREV PHONE NO
SHOULD BE OF LENGTH 10"
                                        else:
                                                if(self.new_phone<=0):
                                                        raise Exception, "NEW PHONE NO
SHOULD BE POSITIVE"
                                                else:
                                                        if(len(str(self.new_phone))!=10):
                                                                raise Exception, "NEW PHONE NO
SHOULD BE OF LENGTH 10"
                                                        else:
                                                                count=0
                                                                for e in
Balance_Enquiry.objects.all():
```

```
                                                                        count=count+1
                                                                for e in
Cash_Withdrawl.objects.all():
                                                                        count=count+1
                                                                for e in Cash_Transfer.objects.all():
                                                                        count=count+1
                                                                for e in Phone_change.objects.all():
                                                                        count=count+1
                                                                for e in Pin_change.objects.all():
                                                                        count=count+1
                                                                self.tid = count+1
                                                                super(Phone_change, self).save()


## pin_change() is the inherited class of trasaction class
## It has 2 fields
## prev_pin ---> Integer
## new_pin ---> Integer
class Pin_change(Transaction):
        prev_pin = models.IntegerField("PREVIOUS PIN")
        new_pin = models.IntegerField("NEW PIN")
        def __unicode__(self):
                return str(self.tid)
        def save(self, *args, **kwargs):
                if(self.tid<=0):
                        raise Exception, "TRANSACTION ID SHOULD BE NON
ZERO POSITIVE"
                else:
                        if(self.rescode<0):
                                raise Exception, "RESPONSE CODE SHOULD BE
POSITIVE"
                        else:
                                if(self.prev_pin<=0):
                                        raise Exception, "PREVIOUS PIN NO SHOULD BE
POSITIVE"
```

```
                    else:
                        if(len(str(self.prev_pin))!=4):
                            raise Exception, "PREVIOUS PIN NO
SHOULD BE OF LENGTH 4"
                        else:
                            if(self.new_pin<=0):
                                raise Exception, "NEW PIN NO
SHOULD BE POSITIVE"
                            else:
                                if(len(str(self.new_pin))!=4):
                                    raise Exception, "NEW PIN NO
SHOULD BE OF LENGTH 4"
                                else:
                                    count=0
                                    for e in
Balance_Enquiry.objects.all():
                                        count=count+1
                                    for e in
Cash_Withdrawl.objects.all():
                                        count=count+1
                                    for e in Cash_Transfer.objects.all():
                                        count=count+1
                                    for e in Phone_change.objects.all():
                                        count=count+1
                                    for e in Pin_change.objects.all():
                                        count=count+1
                                    self.tid = count+1
                                    super(Pin_change, self).save()


## Cash_Transfer() is the inherited class of trasaction class
## It has 3 fields
## ben_acc_num ---> BigInteger
## ben_name ---> Char(100)
## amt_trans ----> Decimal
```

```
class Cash_Transfer(Transaction):
        ben_acc_num = models.BigIntegerField("BENEFICIARY ACCOUNT
NUMBER")
        ben_name = models.CharField("BENEFICIARY NAME",max_length =
100)
        amt_trans =
models.DecimalField("AMOUNT",decimal_places=2,max_digits=10)
        def __unicode__(self):
                return str(self.tid)
        def save(self, *args, **kwargs):
                if(self.tid<=0):
                        raise Exception, "TRANSACTION ID SHOULD BE NON
ZERO POSITIVE"
                else:
                        if(self.rescode<0):
                                raise Exception, "RESPONSE CODE SHOULD BE
POSITIVE"
                        else:
                                if(self.ben_acc_num<0):
                                        raise Exception, "BENEFICIARY ACCOUNT
NUMBER SHOULD BE NON ZERO POSITIVE"
                                else:
                                        if(len(str(self.ben_acc_num))!=12):
                                                raise Exception, "BENEFICIARY ACCOUNT
NUMBER SHOULD BE OF LENGTH 12"
                                        else:
                                                if(self.amt_trans<=0):
                                                        raise Exception, "AMOUNT SHOULD
BE NON ZERO POSITIVE"
                                                else:
                                                        count=0
                                                        for e in Balance_Enquiry.objects.all():
                                                                count=count+1
                                                        for e in Cash_Withdrawl.objects.all():
                                                                count=count+1
                                        for e in Cash_Transfer.objects.all():
                                                count=count+1
                                        for e in Phone_change.objects.all():
                                                count=count+1
                                        for e in Pin_change.objects.all():
                                                count=count+1
                                        self.tid = count+1
                                        super(Cash_Transfer, self).save()


## Cash_Withdrawl() is the inherited class of trasaction class
## It has 2 fields
## amt_with ---> Decimal
## cur_bal ---> Decimal
class Cash_Withdrawl(Transaction):
        amt_with = models.DecimalField("AMOUNT
WITHDRAWN",decimal_places=2,max_digits=10)
        cur_bal = models.DecimalField("CURRENT
BALANCE",decimal_places=2,max_digits=10)
        def __unicode__(self):
                return str(self.tid)
        def save(self, *args, **kwargs):
                if(self.tid<=0):
                        raise Exception, "TRANSACTION ID SHOULD BE NON
ZERO POSITIVE"
                else:
                        if(self.rescode<0):
                                raise Exception, "RESPONSE CODE SHOULD BE
POSITIVE"
                        else:
                                if(self.amt_with<=0):
                                        raise Exception, "AMOUNT WITHDRAWN
SHOULD BE POSITIVE"
                                else:
                                        if(self.cur_bal<0):
```

```
                        raise Exception, "CURRENT BALANCE
SHOULD BE ZERO OR POSITIVE"
                    else:
                        count=0
                        for e in Balance_Enquiry.objects.all():
                            count=count+1
                        for e in Cash_Withdrawl.objects.all():
                            count=count+1
                        for e in Cash_Transfer.objects.all():
                            count=count+1
                        for e in Phone_change.objects.all():
                            count=count+1
                        for e in Pin_change.objects.all():
                            count=count+1
                        self.tid = count+1
                        super(Cash_Withdrawl, self).save()

## Services() is the inherited class of trasaction class
## It has 2 fields
## amount ---> Decimal
## service---> Char
class Services(Transaction):
    amount = models.DecimalField("AMOUNT PAID", decimal_places=2,
max_digits=10)
    service = models.CharField("SERVICE DETAILS", max_length = 100)
    def __unicode__(self):
        return str(self.tid)
from django.test import TestCase, Client
from version1.models import *
import simplejson as json
import datetime
import urllib

class MachineTestCase(TestCase):
    '''Tests for the model Machine'''
    def setUp(self):
        '''Set up fields for the test'''
        self.data_1 = Machine(1,"Delhi", 100.00, 500.00,
datetime.datetime.now(), datetime.datetime.now())
        self.data_1.save()

    def testMachine(self):
        '''Tests if the fields of model 'Machine' are correctly inserted in the
database'''
        self.assertEqual(Machine.objects.all().count(), 1)
        tdata = Machine.objects.get(machine_id = 1)
        self.assertEqual(tdata.location, "Delhi")
        self.assertEqual(tdata.current_balance, 500.00)
        self.assertEqual(tdata.minimum_atm_balance, 100.00)
        self.assertLess(tdata.last_refill_date, datetime.datetime.now())

    def testMachine_negative_money(self):
        '''Asserts that updates of negative values to currency are not
permitted'''
        temp = self.data_1.current_balance
        self.data_1.current_balance = -100
        self.assertRaises(Exception, self.data_1.save)
        self.data_1 = Machine.objects.get(machine_id = 1)
        self.data_1.minimum_atm_balance = -1
        self.assertRaises(Exception, self.data_1.save)
        self.data_1.minimum_atm_balance = temp

    def testMachine_negative_id(self):
        '''Tests negative values for the machine_id field'''
        self.data_1 = Machine.objects.get(machine_id = 1)
        self.data_1.machine_id = -12
        self.assertRaises(Exception, self.data_1.save)

    def testMachine_checkUpdate(self):
        '''Tests if updates to a database entry are done correctly'''
        data = Machine.objects.get(machine_id = 1)
        data.current_balance = 1000.00
        data.save()
        self.assertEqual(Machine.objects.all().count(), 1)

        data_temporary = Machine.objects.get(machine_id = 1)
        self.assertEqual(data_temporary.current_balance, 1000.00)
        data_temporary.minimum_atm_balance = 100.50
        data_temporary.save()
        self.assertEqual(Machine.objects.all().count(), 1)

        data = Machine.objects.get(machine_id = 1)
        self.assertEqual(data.minimum_atm_balance, 100.50)

class AccountExtensionTestCase(TestCase):
    '''Test cases for the Account_Ext model'''
    def setUp(self):
        '''Set up fields for the test'''
        self.data = Account_Ext(123456789012, "M", 9646818259, 100.00)
        self.data.save()

    def testAccountExt(self):
        '''Tests if the fields of model 'Account_Ext' are correctly inserted in
the database'''
        self.assertEqual(Account_Ext.objects.all().count(), 1)
        t = Account_Ext.objects.all()[0]
        self.assertEqual(t.acc_num, 123456789012)
        self.assertEqual(t.name, "M")
        self.assertEqual(t.phone_num, 9646818259)
        self.assertEqual(t.balance, 100.00)

    def testAccountExt_negative_fields(self):
        '''Tests if updates of negative values to different fields are not
permitted'''
        t = Account_Ext.objects.all()[0]
        t.acc_num = 123
        self.assertRaises(Exception, t.save)

        t = Account_Ext.objects.all()[0]
        t.acc_num = -10
        self.assertRaises(Exception, t.save)

        t = Account_Ext.objects.all()[0]
        t.phone_num = -1245465
        self.assertRaises(Exception, t.save)

        t = Account_Ext.objects.all()[0]
        t.phone_num = 1234
        self.assertRaises(Exception, t.save)

        t = Account_Ext.objects.all()[0]
        t.balance = - 10
        self.assertRaises(Exception, t.save)

    def testAccountExt_checkUpdate(self):
        '''Tests if updates to a database entry are done correctly'''
        t = Account_Ext.objects.all()[0]
        t.balance = 12345.00
        t.save()
        self.assertEqual(Account_Ext.objects.all().count(), 1)
        temp = Account_Ext.objects.all()[0]

class ATMCardTestCase(TestCase):
    '''Test cases for the ATM_Card model'''
    def setUp(self):
        '''Set up fields for the test'''
        t = Account_Ext(123456789012, "M", 9646818259, 100.00)
        t.save()
        self.data =
ATM_Card(account_num=Account_Ext.objects.get(acc_num=123456789012),
atmcard_num = 2311, name="M", pin=1234,
date_of_issue=datetime.datetime.now(), expiry_date=datetime.datetime(2012,
11, 10, 12, 00), address="12 B, New Delhi", two_factor=False,
phone_num=9646818259, card_status=True)
        self.data.save()

    def testATMCard(self):
        '''Tests if the fields if modes 'ATM_Card' are correctly inserted in the
database'''
        self.assertEqual(ATM_Card.objects.all().count(), 1)
        card = ATM_Card.objects.all()[0]
        self.assertEqual(card.account_num_id,123456789012)
        self.assertEqual(card.atmcard_num, 2311)
        self.assertEqual(card.phone_num, 9646818259)
        self.assertEqual(card.card_status, True)

    def testATMCard_checkForeignKey(self):
        '''Tests the foreign key constraints for the model'''
        t = ATM_Card.objects.all()[0]
        t.account_num_id = 2
        self.assertRaises(Exception, t.save)

    def testATMCard_checkUpdate(self):
        '''Tests if updates to a database entry are done correctly'''
        t = ATM_Card.objects.all()[0]
        t.pin=1212
        t.save()

        self.assertEqual(ATM_Card.objects.all().count(), 1)
        self.assertEqual(ATM_Card.objects.all()[0].pin, 1212)

    def testATMCard_invalidFields(self):
        '''Tests if wrong values to different fields are not permitted'''
        t = ATM_Card.objects.all()[0]
        t.pin = 12
        self.assertRaises(Exception, t.save)

        self.assertEqual(ATM_Card.objects.all().count(), 1)
        t = ATM_Card.objects.all()[0]
        t.atmcard_num = -1234
        self.assertRaises(Exception, t.save)

        self.assertEqual(ATM_Card.objects.all().count(), 1)
        t = ATM_Card.objects.all()[0]
        t.atmcard_num = 'ABC'
        self.assertRaises(Exception, t.save)

        self.assertEqual(ATM_Card.objects.all().count(), 1)
        t = ATM_Card.objects.all()[0]
        t.phone_num = 1234
        self.assertRaises(Exception, t.save)

class ViewsTestCase(TestCase):
    '''Test cases for views'''
    def setUp(self):
        '''Set up fields for the test'''
        self.url = 'http://localhost:8000/'
        self.client = Client()
        self.machine = Machine(1,"Delhi", 500.00, 10000.00,
datetime.datetime.now(), datetime.datetime(2012, 2, 2))
        self.machine.save()
        self.account_ext = t = Account_Ext(123456789012, "M",
9646818259, 100.00)
        self.account_ext.save()
        self.atmcard = ATM_Card(account_num=self.account_ext,
atmcard_num = 2311, name="M", pin=1234,
date_of_issue=datetime.datetime.now(), expiry_date=datetime.datetime(2012,
11, 10, 12, 00), address="12 B, New Delhi", two_factor=False,
phone_num=9646818259, card_status=True)
        self.atmcard.save()
        def testViews_invalidPages(self):
                '''Tests the response for invalid pages'''
                invalidPages_initial = ['/user/card/', '/user/validatepin',
'/user/validatepasscode', '/user/options', '/user/history', '/user/balanceenquiry',
'/user/cashwithdrawal', '/user/cashtransfer', '/user/pinchange', '/user/phonechange',
'/user/fastcash']
                for page in invalidPages_initial:
                        response = self.client.post(page)
                        self.assertIn(response.status_code, [301, 302])

class APITestCase(TestCase):
        '''Test cases for the API'''
        def setUp(self):
                '''Set up fields for the test'''
                self.url = '/api/v1/'
                self.client = Client()
                self.fields = {'format': 'json'}

        def testAPI_cashwithdrawal(self):
                '''Tests the API for cashwithdrawal'''
                url = self.url + 'cashwithdrawal/'
                res = self.client.get(url + '?' + urllib.urlencode(self.fields))
                response = json.loads(res.content)
                self.assertEqual(response["meta"]["total_count"], 0)

        def testAPI_cashtransfer(self):
                '''Tests the API for cashtransfer'''
                url = self.url + 'cashtransfer/'
                res = self.client.get(url + '?' + urllib.urlencode(self.fields))
                response = json.loads(res.content)
                self.assertEqual(response["meta"]["total_count"], 0)

                Account_Ext(121212121212, "S", 9023043521, 500.00).save()
                acc = Account_Ext(123456789012, "M", 9646818259, 1000.00)
        acc.save()
        atmcard = ATM_Card(account_num=acc, atmcard_num = 2311,
name="M", pin=1234, date_of_issue=datetime.datetime.now(),
expiry_date=datetime.datetime(2012, 11, 10, 12, 00), address="12 B, New
Delhi", two_factor=False, phone_num=9646818259, card_status=True)
        atmcard.save()
        machine = Machine(1,"Delhi", 500.00, 10000.00,
datetime.datetime.now(), datetime.datetime(2012, 2, 2))
        machine.save()
        cash = Cash_Transfer(ben_acc_num=121212121212, ben_name="S",
amt_trans=125.00, tid = 10, rescode = 3, atmcard_num = atmcard, machine_id =
machine, date_time = datetime.datetime.now())
        cash.save()

        res = self.client.get(url + '?' + urllib.urlencode(self.fields))
        response = json.loads(res.content)
        self.assertEqual(response["meta"]["total_count"], 1)
        self.assertEqual(len(response["objects"]), 1)
        self.assertEqual(response["objects"][0]["amt_trans"], 125.00)

    def testAPI_atmcard(self):
        '''Tests the API for atmcard'''
        url = self.url + 'atmcard/'
        res = self.client.get(url + '?' + urllib.urlencode(self.fields))
        response = json.loads(res.content)
        self.assertEqual(response["meta"]["total_count"], 0)

        acc = Account_Ext(123456789012, "M", 9646818259, 1000.00)
        acc.save()
        ATM_Card(account_num=acc, atmcard_num = 2311, name="M",
pin=1234, date_of_issue=datetime.datetime.now(),
expiry_date=datetime.datetime(2012, 11, 10, 12, 00), address="12 B, New
Delhi", two_factor=False, phone_num=9646818259, card_status=True).save()
        res = self.client.get(url + '?' + urllib.urlencode(self.fields))
        response = json.loads(res.content)
```

```
                self.assertEqual(response["meta"]["total_count"], 1)

# Create your views here.
from version1.models import *
from decimal import *
from django.template import RequestContext, loader
from django.http import HttpResponse
from django.shortcuts import render_to_response, get_object_or_404, redirect
import datetime
from django.core.context_processors import csrf
from django.views.decorators.csrf import csrf_protect
import sys
import random
from django.db.models import Avg, Max, Min, Count
from dateutil.relativedelta import relativedelta
import sms
from lxml import etree
import urllib

##To select the machine
##@param: machineid
##@return: is used to redirect to various pages like
'/user/validatepin/','/user/card/','/user/options/'.
@csrf_protect
def main(request):
        if 'cardnumber' in request.session:
                return redirect('/user/validatepin/')
        if 'pinverified' in request.session:
                return redirect('/user/options/')
        machinelist = Machine.objects.all()
        if len(machinelist) > 0:
                machinelistpresent = True
        if request.method == 'POST':
                print request.POST['m']
                request.session['machine'] = request.POST['m']
```

```
            return redirect('/user/card/')
        return render_to_response('finale/main.html',locals())

##for card validation.
##@param: card number
##@return: is used to redirect to various pages like
'/user/validatepin/','/user/card/','/user/'.
@csrf_protect
def index(request):
        if 'machine' not in request.session:
                return redirect('/user/')
        if 'cardnumber' in request.session:
                return redirect('/user/validatepin/')

        if request.method == 'POST':
                cardnum = request.POST['cardnumber']
                try:
                        card = ATM_Card.objects.filter(atmcard_num=cardnum)
                        if not card:
                                cmessage=1
                        elif not card[0].card_status:
                                cmessage=2
                        else:
                                date = datetime.datetime.now()
                                if(card[0].expiry_date < date):
                                        cmessage=3
                                else:
                                        request.session['cardnumber'] = cardnum
                                        return redirect('/user/validatepin/')
                except:
                        e = sys.exc_info()[1]
                        cmessage=1
        return render_to_response('finale/index.html',locals())
```

```
##for pin validation and messaging two factor verification code to registered
phone no. of card holder(if two facto enabled).
##@param: pincode and phoneno(if needed)
##@return: is used to redirect to various pages like
'/user/validatepin/','/user/card/','/user/'  and '/user/options/'.
@csrf_protect
def validatepin(request):
        if 'machine' not in request.session:
                return redirect('/user/')
        if 'cardnumber' not in request.session:
                return redirect('/user/card/')
        if 'pinverified' in request.session:
                return redirect('/user/options/')
        atmcard =
ATM_Card.objects.get(atmcard_num=request.session['cardnumber'])
        username = atmcard.name
        request.session['username'] = username

        if request.method == 'POST':
                try:
                        cardpin = request.POST['pincode']
                        if int(atmcard.pin) == int(cardpin):
                                request.session['pinverified'] = True
                                request.session.set_expiry(300)

                                if atmcard.two_factor:

request.session['passcode']=random.randint(1000,9999)
                                        print request.session['passcode']
                                        sms.sendSMS(atmcard.phone_num,
'PASSCODE:%d' % request.session['passcode'])
                                        return redirect('/user/validatepasscode')
                                return redirect('/user/options')
                        if 'pinattempt' not in request.session:
                                request.session['pinattempt'] = 1
                        else:
                                request.session['pinattempt'] =
request.session['pinattempt'] + 1

                        if request.session['pinattempt'] == 1:
                                        pinmessage = 1
                        elif request.session['pinattempt'] == 2:
                                        pinmessage = 2
                        else:
                                        pinmessage = 3
                                        atmcard.card_status = False
                                        atmcard.save()
                                        request.session.flush()
                except:
                        e = sys.exc_info()[1]
                        if 'pinattempt' not in request.session:
                                request.session['pinattempt'] = 1
                        else:
                                request.session['pinattempt'] =
request.session['pinattempt'] + 1
                        if request.session['pinattempt'] == 1:
                                        pinmessage = 1
                        elif request.session['pinattempt'] == 2:
                                        pinmessage = 2
                        else:
                                        pinmessage = 3
                                        atmcard.card_status = False
                                        atmcard.save()
                                        request.session.flush()
        return render_to_response('finale/pincode.html', locals())

##for passcode validation(if enabled).
##@param: two factor verification code
##@return: is used to redirect to various pages like '/user/card/','/user/'  and
'/user/options/'.
```

```
@csrf_protect
def validatepasscode(request):
        if 'machine' not in request.session:
                return redirect('/user/')
        if 'cardnumber' not in request.session:
                return redirect('/user/card/')
        atmcard =
ATM_Card.objects.get(atmcard_num=request.session['cardnumber'])
        if 'pinverified' in request.session and not atmcard.two_factor:
                return redirect('/user/options/')
        if 'passcodeverified' in request.session:
                return redirect('/user/options/')
        username = atmcard.name
        request.session['username'] = username
        no = atmcard.phone_num
        if request.method == 'POST':
                try:
                        passcode = request.POST['passcode']
                        if int(request.session['passcode']) == int(passcode):
                                request.session['passcodeverified'] = True
                                return redirect('/user/options')
                        if 'passattempt' not in request.session:
                                request.session['passattempt'] = 1
                        else:
                                request.session['passattempt'] =
request.session['passattempt'] + 1

                        if request.session['passattempt'] == 1:
                                        pinmessage = 1
                        elif request.session['passattempt'] == 2:
                                        pinmessage = 2
                        else:
                                        pinmessage = 3
                                        request.session.flush()
                except:
                        e = sys.exc_info()[1]
                        if 'passattempt' not in request.session:
                                request.session['passattempt'] = 1
                        else:
                                request.session['passattempt'] =
request.session['passattempt'] + 1
                        if request.session['passattempt'] == 1:
                                        pinmessage = 1
                        elif request.session['passattempt'] == 2:
                                        pinmessage = 2
                        else:
                                        pinmessage = 3
                                        request.session.flush()
        return render_to_response('finale/passcode.html', locals())

##it shows the various options available to user
##@param: 'request' stores which form method we are using i.e(GET/POST) and
session variable
##@return: is used to redirect to various pages like
'/user/pinvalidation/','/user/card/','/user/'  and '/user/options/'.
@csrf_protect
def options(request):
        if 'machine' not in request.session:
                return redirect('/user/')
        if 'cardnumber' not in request.session:
                return redirect('/user/card/')
        if 'pinverified' not in request.session:
                return redirect('/user/pinvalidation/')
        print request.session.get_expiry_age()
        username = request.session['username']
        return render_to_response('finale/options.html', locals())

##To show the balance to cardholder and saves the transaction in database.
##@param: 'request' stores which form method we are using i.e(GET/POST) and
session variable
```

```
##@return: is used to redirect to various pages like
'/user/validatepin/','/user/card/','/user/'  and '/user/options/'.
@csrf_protect
def balanceenquiry(request):
        if 'machine' not in request.session:
                return redirect('/user/')
        if 'cardnumber' not in request.session:
                return redirect('/user/card/')
        if 'pinverified' not in request.session:
                return redirect('/user/pinvalidation/')
        atmcard =
ATM_Card.objects.get(atmcard_num=request.session['cardnumber'])
        t_acc = Account_Ext.objects.get(acc_num=str(atmcard.account_num))
        t = Balance_Enquiry(atmcard_num = atmcard, machine_id_id =
request.session['machine'],tid = 1,date_time = datetime.datetime.now(),status =
"Completed",rescode = 1,type_trans = "Balance
Enquiry",bal_amount=t_acc.balance)
        t.save()
        bal = t_acc.balance
        username=atmcard.name
        return render_to_response('finale/balance.html', locals())

##To withdraw the money from machine and saves the transaction in database.
##@param: withdrawal amount
##@return: is used to redirect to various pages like
'/user/validatepin/','/user/card/','/user/'  and '/user/options/'.
@csrf_protect
def cashwithdrawal(request):
        if 'machine' not in request.session:
                return redirect('/user/')
        if 'cardnumber' not in request.session:
                return redirect('/user/card/')
        if 'pinverified' not in request.session:
                return redirect('/user/pinvalidation/')
        username=request.session['username']
        if request.method == 'POST':
                try:
                        atmcard =
ATM_Card.objects.get(atmcard_num=request.session['cardnumber'])
                        t_acc =
Account_Ext.objects.get(acc_num=str(atmcard.account_num))
                        if(Decimal(t_acc.balance)>Decimal(request.POST['amount'])):
                                t_acc.balance = t_acc.balance -
Decimal(request.POST['amount'])
                                t_acc.save()
                                t = Cash_Withdrawl(atmcard_num = atmcard,
machine_id_id = request.session['machine'],tid = 1,date_time =
datetime.datetime.now(),status = "Completed",rescode = 2,type_trans = "Cash
Withdrawal",amt_with =
Decimal(request.POST['amount']),cur_bal=t_acc.balance)
                                t.save()
                                wdmessage = 1
                        else:
                                ta = Cash_Withdrawl(atmcard_num = atmcard,
machine_id_id =request.session['machine'], tid = 1, date_time =
datetime.datetime.now(),status = "Not Completed",rescode = 11,type_trans =
"Cash Withdrawal",amt_with = Decimal(request.POST['amount']),cur_bal =
t_acc.balance)
                                ta.save()
                                wdmessage = 2
                except:
                        e = sys.exc_info()[1]
                        wdmessage=3
        return render_to_response('finale/cashwithdrawal.html', locals())

##To transfer the cash from cardholder's account to another specified account and
saves the transaction in database.
##@param: name and account no. in which we want to transfer the money and
amount to transfer.
```

```
##@return: is used to redirect to various pages like
'/user/pinvalidation/','/user/card/','/user/'  and '/user/options/'.
@csrf_protect
def cashtransfer(request):
        if 'machine' not in request.session:
                return redirect('/user/')
        if 'cardnumber' not in request.session:
                return redirect('/user/card/')
        if 'pinverified' not in request.session:
                return redirect('/user/pinvalidation/')
        username=request.session['username']
        if request.method == 'POST':
                try:
                        accnum = request.POST['acc_num']
                        accname = request.POST['name']
                        acc_2 =
Account_Ext.objects.filter(acc_num=accnum,name=accname)
                        if not acc_2:
                                ctmessage = 0
                        else:
                                atmcard =
ATM_Card.objects.get(atmcard_num=request.session['cardnumber'])
                                t_acc =
Account_Ext.objects.get(acc_num=str(atmcard.account_num))

if(Decimal(t_acc.balance)>Decimal(request.POST['amount'])):
                                        t_acc.balance = t_acc.balance -
Decimal(request.POST['amount'])
                                        t_acc.save()
                                        acc_2[0].balance = acc_2[0].balance +
Decimal(request.POST['amount'])
                                        acc_2[0].save()
                                        t = Cash_Transfer(atmcard_num = atmcard,
machine_id_id = request.session['machine'],tid = 1,date_time =
datetime.datetime.now(),status = "Completed",rescode = 3,type_trans = "Cash
Transfer",amt_trans =
Decimal(request.POST['amount']),ben_acc_num=accnum,ben_name=accname)
                                        t.save()
                                        ctmessage = 1
                                else:
                                        ta = Cash_Transfer(atmcard_num = atmcard,
machine_id_id = request.session['machine'], tid = 1, date_time =
datetime.datetime.now(),status = "Not Completed",rescode = 12,type_trans =
"Cash Transfer",amt_trans =
Decimal(request.POST['amount']),ben_acc_num=accnum,ben_name=accname)
                                        ta.save()
                                        ctmessage = 2
                except:
                        e = sys.exc_info()[1]
                        ctmessage=3
        return render_to_response('finale/cashtransfer.html', locals())

##To change the pin and saves the transaction in database.
##@param: previous pincode,new pincode,confirm pincode
##@return: is used to redirect to various pages like
'/user/validatepin/','/user/card/','/user/'  and '/user/options/'.
@csrf_protect
def pinchange(request):
        if 'machine' not in request.session:
                return redirect('/user/')
        if 'cardnumber' not in request.session:
                return redirect('/user/card/')
        if 'pinverified' not in request.session:
                return redirect('/user/pinvalidation/')
        username = request.session['username']
        if request.method == 'POST':
                try:
                        atmcard =
ATM_Card.objects.get(atmcard_num=request.session['cardnumber'])
                        cardpin = request.POST['pincode']
                        npin = request.POST['npincode']
                        cpin = request.POST['cpincode']
                        if int(atmcard.pin) == int(cardpin):
                                if int(npin) == int(cpin):
                                        if int(atmcard.pin) == int(cpin):
                                                #same no need to do any thing
                                                pcmessage=1
                                        else:
                                                #successfull
                                                pcmessage=4
                                                t = Pin_change(atmcard_num = atmcard,
machine_id_id = request.session['machine'],tid = 1,date_time =
datetime.datetime.now(),status = "Completed",rescode = 4,type_trans = "Pin
Change",prev_pin=cardpin,new_pin=npin)
                                                t.save()
                                                atmcard.pin=int(cpin)
                                                atmcard.save()

                                else:
                                        #new pin and confirm pin are different
                                        pcmessage=2

                        else:
                                #invalid pincode
                                pcmessage=3
                except:
                        e = sys.exc_info()[1]
                        pcmessage=5
        return render_to_response('finale/pinchange.html', locals())

##To change the phoneno. and saves the transaction in database.
##@param: new phoneno and confirm phoneno
##@return: is used to redirect to various pages like
'/user/validatepin/','/user/card/','/user/'  and '/user/options/'.
@csrf_protect
```

```
def phonechange(request):
        if 'machine' not in request.session:
                return redirect('/user/')
        if 'cardnumber' not in request.session:
                return redirect('/user/card/')
        if 'pinverified' not in request.session:
                return redirect('/user/pinvalidation/')
        username = request.session['username']
        if request.method == 'POST':
                try:
                        atmcard =
ATM_Card.objects.get(atmcard_num=request.session['cardnumber'])
                        nphone = request.POST['nphone']
                        cphone = request.POST['cphone']
                        if int(nphone) == int(cphone):
                                        if int(atmcard.phone_num) == int(cphone):
                                                #same no need to do any thing
                                                pcmessage=1
                                        else:
                                                #successfull
                                                pcmessage=4
                                                t = Phone_change(atmcard_num = atmcard,
machine_id_id = request.session['machine'],tid = 1,date_time =
datetime.datetime.now(),status = "Completed",rescode = 4,type_trans = "Pin
Change",prev_phone=atmcard.phone_num,new_phone=nphone)
                                                t.save()
                                                atmcard.phone_num=int(cphone)
                                                atmcard.save()

                        else:
                                #new phoneno and confirm phoneno are different
                                pcmessage=2
                except:
                        e = sys.exc_info()[1]
                        pcmessage=5
        return render_to_response('finale/phonechange.html', locals())

##To withdraw the money by just selecting one of the option available to
cardholder and saves the transaction in database.
##@param: selected amount
##@return: is used to redirect to various pages like
'/user/validatepin/','/user/card/','/user/'  and '/user/options/'.
@csrf_protect
def fastcash(request):
        if 'machine' not in request.session:
                return redirect('/user/')
        if 'cardnumber' not in request.session:
                return redirect('/user/card/')
        if 'pinverified' not in request.session:
                return redirect('/user/pinvalidation/')
        username=request.session['username']
        if request.method == 'POST':
                try:
                        atmcard =
ATM_Card.objects.get(atmcard_num=request.session['cardnumber'])
                        t_acc =
Account_Ext.objects.get(acc_num=str(atmcard.account_num))
                        if(Decimal(t_acc.balance)>Decimal(request.POST['fcash'])):
                                t_acc.balance = t_acc.balance -
Decimal(request.POST['fcash'])
                                t_acc.save()
                                t = Cash_Withdrawl(atmcard_num = atmcard,
machine_id_id = request.session['machine'],tid = 1,date_time =
datetime.datetime.now(),status = "Completed",rescode = 2,type_trans = "Cash
Withdrawal",amt_with = Decimal(request.POST['fcash']),cur_bal=t_acc.balance)
                                t.save()
                                fcmessage = 1
                                request.session.flush()
                        else:
```

```
                                        ta = Cash_Withdrawl(atmcard_num_id =
request.session['cardnumber'], machine_id_id =request.session['machine'], tid =
1, date_time = datetime.datetime.now(),status = "Not Completed",rescode =
11,type_trans = "Cash Withdrawal",amt_with =
Decimal(request.POST['fcash']),cur_bal = t_acc.balance)
                                        ta.save()
                                        fcmessage = 2
                except:
                        e = sys.exc_info()[1]
                        fcmessage=3
        return render_to_response('finale/fastcash.html', locals())

##To show the history of successfull transactions(only involving money) made
by cardholder and saves the transaction in database.
##@param: 'request' stores which form method we are using i.e(GET/POST) and
session variable
##@return: is used to redirect to various pages like
'/user/validatepin/','/user/card/','/user/'  and '/user/options/'.
@csrf_protect
def history(request):
        if 'machine' not in request.session:
                return redirect('/user/')
        if 'cardnumber' not in request.session:
                return redirect('/user/card/')
        if 'pinverified' not in request.session:
                return redirect('/user/pinvalidation/')
        username = request.session['username']
        cardnum=request.session['cardnumber']
        cashw=Cash_Withdrawl.objects.filter(atmcard_num=cardnum,rescode=2)
        casht=Cash_Transfer.objects.filter(atmcard_num=cardnum,rescode=3)
        list_transaction=[]

        for e in cashw:
                dict={}
                dict['date'] = e.date_time
```

```
            dict['tt'] = e.type_trans
            dict['amount'] =e.amt_with
            list_transaction.append(dict)
        for e in casht:
            dict={}
            dict['date'] = e.date_time
            dict['tt'] = e.type_trans
            dict['amount'] =e.amt_trans
            list_transaction.append(dict)
        return render_to_response('finale/history.html', locals())

def services_mock_listBiller(request):
        r = etree.Element('BankServices')
        billers = etree.SubElement(r, 'Billers')

        b1 = etree.SubElement(billers, 'Biller', category="MutualFunds")
        etree.SubElement(b1, 'name').text = "LIC"
        etree.SubElement(b1, 'account_id').text = "201"

        b1 = etree.SubElement(billers, 'Biller', category="Water")
        etree.SubElement(b1, 'name').text = "Delhi Jal Board"
        etree.SubElement(b1, 'account_id').text = "202"

        b1 = etree.SubElement(billers, 'Biller', category="Telephone")
        etree.SubElement(b1, 'name').text = "Airtel"
        etree.SubElement(b1, 'account_id').text = "203"

        b1 = etree.SubElement(billers, 'Biller', category="Telephone")
        etree.SubElement(b1, 'name').text = "MTNL"
        etree.SubElement(b1, 'account_id').text = "204"

        b1 = etree.SubElement(billers, 'Biller', category="Exam")
        etree.SubElement(b1, 'name').text = "GATE"
        etree.SubElement(b1, 'account_id').text = "205"
        b1 = etree.SubElement(billers, 'Biller', category="Electricity")
        etree.SubElement(b1, 'name').text = "Delhi Electricity Board"
        etree.SubElement(b1, 'account_id').text = "206"

        b1 = etree.SubElement(billers, 'Biller', category="Exam")
        etree.SubElement(b1, 'name').text = "IIT JEE"
        etree.SubElement(b1, 'account_id').text = "207"

        return HttpResponse(etree.tostring(r), mimetype="text/xml")

@csrf_protect
def services(request):
        if 'machine' not in request.session:
                return redirect('/user/')
        if 'cardnumber' not in request.session:
                return redirect('/user/card/')
        if 'pinverified' not in request.session:
                return redirect('/user/pinvalidation/')
        r = urllib.urlopen('http://localhost:8000/BankServices/rest/biller')
        print r.read()
        serv = etree.parse(r.read())
        serviceslist = root.iter()
        return render_to_response('finale/services.html', locals())

##To delete the session.
##@param: 'request' stores which form method we are using i.e(GET/POST) and
session variable
##@return: is used to redirect to various pages like
'/user/pinvalidation/','/user/card/' and '/user/'.
@csrf_protect
def exit(request):
        if 'machine' not in request.session:
                return redirect('/user/')
        if 'cardnumber' not in request.session:
                return redirect('/user/card/')
        if 'pinverified' not in request.session:
                return redirect('/user/pinvalidation/')
        request.session.flush()
        return redirect('/user/')

## admin_index() displays the index page of the administrator where he will be
asked to fill the user name and password
@csrf_protect
def admin_index(request):
        return render_to_response('admin_user/index.html', locals())

## admin_verify_user() user will verify the administrator and password entered
by him
## this function also starts the session for the Administrator if username and
password are entered correctly
##@param: username and password of admin
@csrf_protect
def admin_verify_user(request):
        if request.method == 'POST':

admin=Admin.objects.filter(Admin_id=request.POST['username'],Password=req
uest.POST['password'])
                if not admin:
                        failed=True
                        return render_to_response('admin_user/index.html',
{"failed":failed})#locals())
                else:
                        request.session['login'] = True
                        return
render_to_response('admin_user/main_page.html',locals())
        else:
                return redirect('/admin_user/')

## admin_main_page() displays all the option for the administrator
def admin_main_page(request):
        if 'login' not in request.session:
                return render_to_response('admin_user/index.html', locals())
        else:
                return render_to_response('admin_user/main_page.html', locals())


## admin_add_card() displays the page which asks for card details
def admin_add_card(request):
        if 'login' not in request.session:
                return render_to_response('admin_user/index.html', locals())
        else:
                acc_list = Account_Ext.objects.all()
                return
render_to_response('admin_user/add_atm.html',{"acc_list":acc_list})

## admin_add_card_operation() creates new ATM card
##@param: account no.,phoneno.,name,address,twofactorverification
def admin_add_card_operation(request):
        if 'login' not in request.session:
                return render_to_response('admin_user/index.html', locals())
        else:
                try:
                        if ((request.GET['phone']=="") or
(request.GET['address']=="")or (request.GET['pin']=="") or
(request.GET['name']=="")):
                                empty=True
                                return
render_to_response('admin_user/view_atm_status.html',
{"Machine_list":Machine_list}, {"empty":empty})
                        else:
                                if (request.GET['two']=='False'):
                                        t=False
                                else:
                                        t=True
                                p=datetime.datetime.now()
                        q =p + relativedelta(years=+4)
                        e =
ATM_Card(atmcard_num=(1111+random.randint(1000,8888)),account_num_id=
request.GET['acc'],name=request.GET['name'],pin=request.GET['pin'],date_of_is
sue=p,expiry_date=q,address=request.GET['address'],two_factor=t,phone_num=r
equest.GET['phone'],card_status=True)
                        e.save()
                        return
render_to_response('admin_user/main_page.html',locals())
            except Exception as e:
                        acc_list = Account_Ext.objects.all()
                        m=True
                        #return: render_to_response('admin_user/index.html',
locals())
                        return
render_to_response('admin_user/add_atm.html',locals())

# admin_atm_status() will show the details of all the ATM-machines
def admin_atm_status(request):
      if 'login' not in request.session:
            return render_to_response('admin_user/index.html', locals())
      else:
            Machine_list = Machine.objects.all()
            return render_to_response('admin_user/view_atm_status.html',
{"Machine_list":Machine_list})

# admin_update_refill() will send a request to the autority to fill the refill of that
ATM machine
def admin_update_refill(request):
      if 'login' not in request.session:
            return render_to_response('admin_user/index.html', locals())
      else:
            [machine] =Machine.objects.filter(machine_id=request.GET['id'])
            date_time = datetime.datetime.now()
            machine.next_maintainence_date=date_time
```

```
                machine.save()
                Machine_list = Machine.objects.all()
                return render_to_response('admin_user/view_atm_status.html',
{"Machine_list":Machine_list})

## admin_update_card_details will ask the administrator to enter the atm card
number which he want to update
def admin_update_card_details(request):
        if 'login' not in request.session:
                return render_to_response('admin_user/index.html', locals())
        else:
                return render_to_response('admin_user/enter_card_no.html', locals())


## admin_card_validation() will validate the atm card no and it is correct then it
will take the administrator to options else it will dispay that card is not valid
##@param: cardnumber
def admin_card_validation(request):
        if 'login' not in request.session:
                return render_to_response('admin_user/index.html', locals())
        else:
                try:
                        if request.method == 'POST':
                                [cardcheck]
=ATM_Card.objects.filter(atmcard_num=request.POST['cardnumber'])
                                if not cardcheck:
                                        failed=True
                                        return
render_to_response('admin_user/enter_card_no.html', {"failed":failed})
                                else:
                                        #card=cardcheck
                                        request.session['admin_session_card'] =
request.POST['cardnumber']
                                        request.session['admin_session']=1
                                        return
render_to_response('admin_user/update_card_details.html', locals())
                except Exception as e:
                                m=True
                                return
render_to_response('admin_user/enter_card_no.html', {"m":m})

## admin_update_card_main_page() displays all the options on the screen for the
administrator too update the card details
def     admin_update_card_main_page(request):
        if 'login' not in request.session:
                return render_to_response('admin_user/index.html', locals())
        else:
                if 'admin_session' not in request.session:
                        return render_to_response('admin_user/enter_card_no.html',
locals())
                else:
                        return
render_to_response('admin_user/update_card_details.html', locals())

## admin_block_card() displays the page on which we can block the card

def admin_block_card(request):
        if 'login' not in request.session:
                return render_to_response('admin_user/index.html', locals())
        else:
                if 'admin_session' not in request.session:
                        return render_to_response('admin_user/enter_card_no.html',
locals())
                else:
                        return render_to_response('admin_user/block_card.html',
locals())

## admin_block_card() blocks the ATM card and saves in the database.
def admin_block_card_operation(request):
        if 'login' not in request.session:
                return render_to_response('admin_user/index.html', locals())
        else:
                if 'admin_session' not in request.session:
                        return render_to_response('admin_user/enter_card_no.html',
locals())
                else:
                        [cardcheck]
=ATM_Card.objects.filter(atmcard_num=request.session['admin_session_card'])
                        cardcheck.card_status=False
                        cardcheck.save()
                        return
render_to_response('admin_user/update_card_details.html', locals())

## admin_activate_card() displays the page which asks for confirmation of the
administrator to activate the ATM card
def admin_activate_card(request):
        if 'login' not in request.session:
                return render_to_response('admin_user/index.html', locals())
        else:
                if 'admin_session' not in request.session:
                        return render_to_response('admin_user/enter_card_no.html',
locals())
                else:
                        return render_to_response('admin_user/activate_card.html',
locals())

## admin_activates_card() activates the ATM card by changing status to true in
database
def admin_activate_card_operation(request):
        if 'login' not in request.session:
                return render_to_response('admin_user/index.html', locals())
        else:
                if 'admin_session' not in request.session:
```

```
                return render_to_response('admin_user/enter_card_no.html',
locals())
            else:
                [cardcheck]
=ATM_Card.objects.filter(atmcard_num=request.session['admin_session_card'])
                cardcheck.card_status=True
                cardcheck.save()
                return
render_to_response('admin_user/update_card_details.html', locals())

## admin_reset_pincode() displays the page which aks the administrator to enter
the new pin
def admin_reset_pincode(request):
        if 'login' not in request.session:
            return render_to_response('admin_user/index.html', locals())
        else:
            if 'admin_session' not in request.session:
                return render_to_response('admin_user/enter_card_no.html',
locals())
            else:
                return render_to_response('admin_user/reset_pin.html', locals())



## admin_reset_phone() displays the page from which adminstrator can change
the pincode of cardholder
##@param: new pincode and confirm pincode
def admin_reset_pincode_operation(request):
        if 'login' not in request.session:
            return render_to_response('admin_user/index.html', locals())
        else:
            if 'admin_session' not in request.session:
                return render_to_response('admin_user/enter_card_no.html',
locals())
            else:
```

```
                        try:
                                if ((request.GET['password1']=="") or
(request.GET['password2']=="")):
                                        empty=True
                                        return
render_to_response('admin_user/reset_pin.html', {"empty":empty})
                                else:
                                        if
(request.GET['password1']==request.GET['password2']):
                                                [cardcheck]
=ATM_Card.objects.filter(atmcard_num=request.session['admin_session_card'])
                                                cardcheck.pin=request.GET['password1']
                                                cardcheck.save()
                                                return
render_to_response('admin_user/update_card_details.html', locals())
                                        else:
                                                match=True
                                                return
render_to_response('admin_user/reset_pin.html', {"match":match})
                        except Exception as e:
                                m=True
                                return render_to_response('admin_user/reset_pin.html',
{"m":m})

## admin_reset_phone() displays the page from which adminstrator can change
the phone
def admin_reset_phone(request):
        if 'login' not in request.session:
                return render_to_response('admin_user/index.html', locals())
        else:
                if 'admin_session' not in request.session:
                        return render_to_response('admin_user/enter_card_no.html',
locals())
                else:
```

```
                return render_to_response('admin_user/reset_phone.html',
locals())

## admin_reset_phone_operation() reset the phone to new phone enetered by the
administrator
##@param: new phoneno, confirm phoneno.
def admin_reset_phone_operation(request):
        if 'login' not in request.session:
                return render_to_response('admin_user/index.html', locals())
        else:
                if 'admin_session' not in request.session:
                        return render_to_response('admin_user/enter_card_no.html',
locals())
                else:
                        try:
                                if ((request.GET['phone1']=="") or
(request.GET['phone2']=="")):
                                        empty=True
                                        return
render_to_response('admin_user/reset_phone.html', {"empty":empty})
                                else:
                                        if (request.GET['phone1']==request.GET['phone2']):

                                                [cardcheck]
=ATM_Card.objects.filter(atmcard_num=request.session['admin_session_card'])
                                                cardcheck.phone_num=request.GET['phone1']
                                                cardcheck.save()
                                                return
render_to_response('admin_user/update_card_details.html', locals())
                                        else:
                                                match=True
                                                return
render_to_response('admin_user/reset_phone.html', {"match":match})
                        except Exception as e:
                                m=True
```

```
                    return render_to_response('admin_user/reset_phone.html',
{"m":m})

## admin_view_history() shows the previous history of the ATM card

def admin_view_history(request):

    if 'login' not in request.session:
        return render_to_response('admin_user/index.html', locals())
    else:
        if 'admin_session' not in request.session:
            return render_to_response('admin_user/enter_card_no.html',
locals())
        else:
            cardnum=request.session['admin_session_card']
            cashw=Cash_Withdrawl.objects.filter(atmcard_num=cardnum)
            print cashw
            casht=Cash_Transfer.objects.filter(atmcard_num=cardnum)
            print casht
            bal = Balance_Enquiry.objects.filter(atmcard_num=cardnum)
            print bal
            pin = Pin_change.objects.filter(atmcard_num=cardnum)
            print pin
            phone = Phone_change.objects.filter(atmcard_num=cardnum)
            print phone
            list_transaction=[]

            for e in cashw:
                dict={}
                dict['date'] = e.date_time
                dict['tt'] = e.type_trans
                dict['amount'] = e.amt_with
                dict['status'] = e.status
                list_transaction.append(dict)
            for e in casht:
                dict={}
                dict['date'] = e.date_time
                dict['tt'] = e.type_trans
                dict['amount'] =e.amt_trans
                dict['status'] = e.status
                list_transaction.append(dict)
            for e in bal:
                dict={}
                dict['date'] = e.date_time
                dict['tt'] = e.type_trans
                dict['amount'] = 'NOT APLLICABLE'
                dict['status'] = e.status
                list_transaction.append(dict)
            for e in pin:
                dict={}
                dict['date'] = e.date_time
                dict['tt'] = e.type_trans
                dict['amount'] = 'NOT APLLICABLE'
                dict['status'] = e.status
                list_transaction.append(dict)
            for e in phone:
                dict={}
                dict['date'] = e.date_time
                dict['tt'] = e.type_trans
                dict['amount'] = 'NOT APLLICABLE'
                dict['status'] = e.status
                list_transaction.append(dict)

            return render_to_response('admin_user/view_history.html',
locals())

##admin_update_date() displays the page from which admin can update the
expiry date of card holder
def admin_update_date(request):
    if 'login' not in request.session:
            return render_to_response('admin_user/index.html', locals())
        else:
            if 'admin_session' not in request.session:
                    return render_to_response('admin_user/enter_card_no.html',
locals())
            else:
                    return
render_to_response('admin_user/update_expiring_date.html', locals())

## admin_update_date_operation() update the expiring date in the database by 4
years
def admin_update_date_operation(request):

        if 'login' not in request.session:
            return render_to_response('admin_user/index.html', locals())
        else:
            if 'admin_session' not in request.session:
                    return render_to_response('admin_user/enter_card_no.html',
locals())
            else:
                    [cardcheck]
=ATM_Card.objects.filter(atmcard_num=request.session['admin_session_card'])
                    t=cardcheck.expiry_date
                    t += relativedelta(years=+4)
                    cardcheck.expiry_date=t
                    cardcheck.save()
                    return
render_to_response('admin_user/update_card_details.html', locals())

##admin_logout() logsout the administrator and deletes the session.
def admin_logout(request):
        for sesskey in request.session.keys():
            del request.session[sesskey]
        return render_to_response('admin_user/index.html', locals())
```

CHAPTER 5

MENU- EXPLANATION

Menu Explanation:-

This section refers to the various types of interfaces which the user has to face during operating the computerized system of "System".

The section refers with the entire interface [Screens] a user will have to face while operating the current system. It shows the various screens appearing for different transactions. All the screens of different transactions in the system are shown here.

Menus and links of ATM System are as follows;

- Welcome: - This is main file of ATM system, by using this we can enter to the ATM system & go to next screen as ATM card no.
- ATM card no: - by using this we can enter the ATM card no and PIN no & these are correct then we go to next screen as Account type.
- Account Type: - by using this we can press our account type if account type is match then go to next screen as Transaction. If account type is not match then display appropriate message and go to welcome screen.
- Transaction Menu: - by using this we can go to various screens such as cash withdrawal, cash deposit, balance enquiry, mini statement, pin change, loan information and help.
- Cash Withdrawal: - by using this we withdraw cash from our account.
- Cash Deposit: - by using this we deposit cash to our account.
- Balance Enquiry: - by using this we can see our available balance.
- Mini Statement: - by using this we can see our last 10 transaction.
- PIN Change: - by using this we can change or update our PIN no of ATM card.
- Loan Information: - by using this we can see various loan rates such as home loan, car loan and personal loan.
- Help: - by using this we can see how to operate existing system.

Drawbacks:-

- ATM System requires 24 hours security therefore it requires security guards.
- ATM System requires small shop to store machine therefore need to pay that shop rents.
- If money in the ATM is not available then it takes some time to fill the money in the cash stock box in ATM machine.
- If ATM card is lost & this lost ATM card uses any other person then it will become dangerous.

Limitations:-

- Our system may become obsolete as in computer industry; technological developments are very fast, new software, new utilities may obsolete this system.
- System security is ATM card no and PIN no dependent, if security about ATM card no and PIN no information is not maintained, system could be in great danger.
- This system is constructed and developed for text environment so pack gives best appearance and performance under text environment but poor appearance on GUI environment.
- System requires electricity to function; absence may result in chaotic situation in the organizational procedures.

Proposed enhancement:-

The system is designed keeping in mind the current requirements of the ATM. However some aspects were not considered and system can easily changing where shop requirements are changed.
Some of the enhancements can be:

- System can be design in GUI environment.
- System can be design to work without manpower by using Scratch card.
- The system can be made flexible so that new modules can be added at any given time.

- In future system can be construct the modules of fund transfer, mobile recharge, pay electricity bill can be developed.

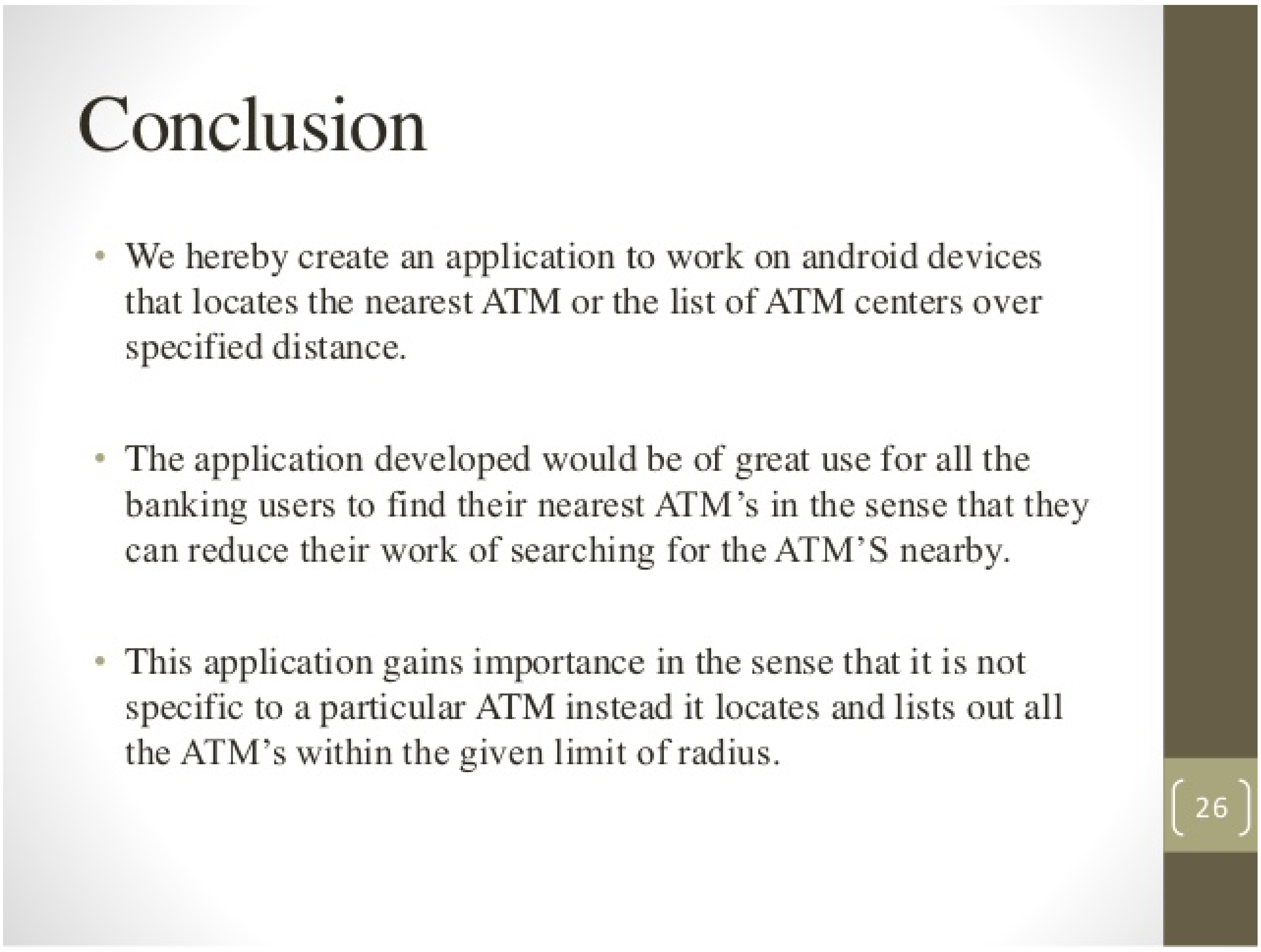
Abbreviations:-

After we have completed the project we are sure that problem in the existing system would overcome. "**Automated Teller Machine (ATM). System"** process has been computerized reduced human error and to increase the efficiency. The main focus of this project is to lessen human efforts. The maintenance of record is made efficient, as all the records are stored in the database through which the data can be retrieved easily. The navigation control is provided in all form to navigate through large amount of records. Our main aim of the project is to provide correct banking services to customer of the bank at any time any place.

The problem which exited in earlier system, have been removed to large extent. The computerization of the "**Automated Teller Machine (ATM) System."** will not only improve the efficiency but will also reduce human stress thereby indirectly improving human resources.

CHAPTER 6

CONCLUSION



Conclusion

- We hereby create an application to work on android devices that locates the nearest ATM or the list of ATM centers over specified distance.
- The application developed would be of great use for all the banking users to find their nearest ATM's in the sense that they can reduce their work of searching for the ATM'S nearby.
- This application gains importance in the sense that it is not specific to a particular ATM instead it locates and lists out all the ATM's within the given limit of radius.

26

CHAPTER 7

REFERENCE

REFERENCE:

References & Bibliography:-

Material referred for the development of this "**Automated Teller Machine (ATM) System**" is as follows.

Bibliography:-

Books referred during project development**:**

- Head-First Python : Paul Barry
- Learn Python the Hard Way : Zed A. Shaw
- A Byte of Python : C.H. Swaroop

Website Reference:-

- www.Pythonworld.com
- www.wikipedia.com

THANK YOU.