CPD

Banking application

**VERSION 1.0**

APPROVAL SIGNATURES

| CONTRACTOR | | DATE |
| --- | --- | --- |
| <<Deliverable Owner>>  <<John Doe, Manager>> | <<Signature>> |  |
| <<Vendor Project Manager>> | <<Signature>> |  |
| <<Vendor Managing Director>> | <<Signature>> |  |

| **PROJECT MANAGER** | | | | |
| --- | --- | --- | --- | --- |
| Date: | 02.04.2025 | To: | **ABC,** Project Manager | |
|  | I approve this deliverable and have no further questions or comments. | | | |
|  | I approve this deliverable conditionally, contingent on the review and approval of the following corrections (see comments). | | | |
|  | I reject this deliverable for the following reasons identified (see comments). | | | |
|  | | | | |
| **<<SIGNATURE>>** | | | | **<<DATE>>** |
| Comments | | | | |
|  | | | | |

DOCUMENT HISTORY

| **DOCUMENT APPROVAL HISTORY** | |
| --- | --- |
| Prepared By |  |
| Reviewed By |  |
| Approved By |  |

| **DOCUMENT REVISION HISTORY** | | | |
| --- | --- | --- | --- |
| Date | Document Version | Revision Description | Author |
| 12/11/2024 | 1.0 | Initial Version |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

TABLE OF CONTENTS

1. Introduction 1

1.1. Purpose of this Document 1

1.2. Scope of this Document……………………………………………………………………………1-2

1.3. Overview 1-2

2. GenERAL Description 3

3. FUNCTional Requirements 4

3.1 User Functionalities

3.2 Admin Functionalities

4. system design 5

4.1 Architecture

4.2 Database Schema

4.3 Use Case Diagram

5. Development plan 6

5.1 Risks and Mitigation

6. Non-functional attributes 7

7. flow chart 8

8. code…………………………. 9-15

# Introduction

The banking industry has evolved significantly with **digital transformation**, requiring secure, efficient, and scalable software solutions. This seminar focuses on the **development of a banking application**, following the **Software Development Life Cycle (SDLC)** stages.

* **Requirement Analysis** – Understanding user needs like account management, transactions, and security.
* **System Design** – Structuring the database, defining workflows, and designing architecture.
* **Development & Implementation** – Writing code for user authentication, transactions, and admin controls.
* **Testing & Debugging** – Ensuring secure and error-free operations.
* **Deployment & Maintenance** – Launching and continuously improving the application.

**1****.1 Purpose of This Document**

This document provides a C-based implementation of a Banking Software System that allows users to manage their bank accounts securely. The purpose of this project is to:

1. Enable Users to:
   * Register and log in securely.
   * Check account details, including balance.
   * Transfer money between accounts.
2. Enable Admin to:
   * Create and delete user accounts.
   * Manage funds (credit/debit).
   * Reset user passwords.
3. Implement Banking Features Securely:
   * User authentication (username & password).
   * Basic transaction handling (fund transfers).
   * Admin-controlled account management.

**1.2 Objective:**

To develop a **secure and efficient Banking Application** that enables:

* Users to **create accounts, check balances, and transfer money**.
* An **admin to manage user accounts** (create, delete, credit/debit funds, reset passwords).

**Target Users:**

* **End-users:** Customers who need online banking services.
* **Admin:** A bank administrator who manages user accounts.

### ****1.3 Overview of the Project****

This project is a **Banking Application** implemented in **C** that allows users to securely manage their bank accounts while providing an admin with control over account operations. The system enables users to **log in, check their account balance, and transfer money** to other accounts. The **admin has privileged access** to create and delete user accounts, credit or debit funds, and reset user passwords. The project follows the **Software Development Life Cycle (SDLC)** stages, including **requirement analysis, system design, development, testing, and deployment.** It ensures **data security, user authentication, and transaction management** while maintaining a simple and efficient structure. The system can be further enhanced with **file handling, encryption, and database integration** for improved scalability and security.

**2. General Description**

The Banking Application is a C-based program designed to provide basic banking functionalities for both users and administrators. The system allows users to create accounts, log in securely, check account details, and transfer money between accounts. An admin has the ability to create or delete user accounts, credit or debit balances, and reset user passwords. The program follows a menu-driven approach, enabling easy navigation for users and administrators.

The project is structured with secure authentication mechanisms, ensuring only registered users can access their accounts. It uses arrays to store user data, with unique account numbers assigned to each user. The system follows a modular programming approach, making it easy to extend and integrate with additional features such as file handling for data persistence, encryption for security, and database connectivity for scalability.

**3. Functional Requirements:**

**3.1 User Functionalities:**

* + User registration with **username, password, email, mobile number**.
  + Login with authentication (username/password).
  + View account details (account number, balance).
  + Transfer money to another account.
  + Receive money and view transaction history.

**3.2 Admin Functionalities:**

* + Login with admin credentials.
  + Create and delete user accounts.
  + Assign account numbers to users.
  + Credit/Debit funds from user accounts.
  + Reset user passwords.

**4. System Design**

**4.1 Architecture:**

* **Frontend:** C-based CLI (Command Line Interface).
* **Backend:** File-based storage (or database if extended).
* **Data Storage:** Structured using **C structs and file handling**.

**4.2 Database Schema:**

| **Table** | **Fields** |
| --- | --- |
| Users | ID, Username, Password, Mobile, Email, Account Number, Balance |
| Transactions | Transaction ID, Sender ID, Receiver ID, Amount, Date |

**4.3 Use Case Diagram:**

* **User**: Login → View Account → Transfer Money
* **Admin**: Login → Create User → Manage Accounts → Logout

**5. Development Plan**

| **Stage** | **Tasks** | **Timeline** |
| --- | --- | --- |
| **Requirement Analysis** | Gather user/admin needs, define scope | **Day 1** |
| **System Design** | Design database schema, flowcharts | **Day 2** |
| **Development** | Implement login, transactions, admin panel | **Day 3-4** |
| **Testing** | Test cases, bug fixes | **Day 5** |
| **Deployment** | Host and optimize code | **Day 6** |
| **Documentation** | Create user/admin guides | **Day 7** |

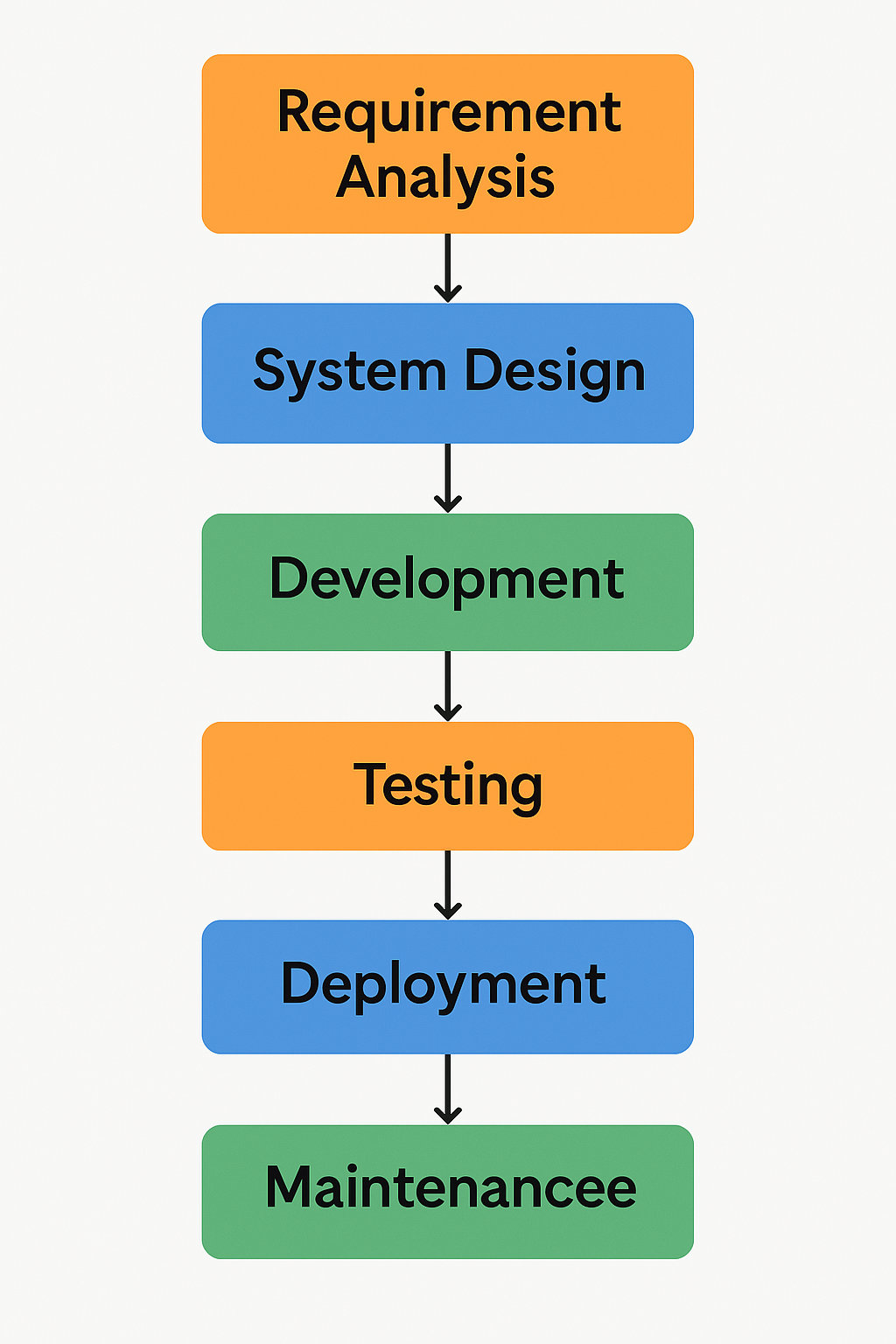
**5.1 Risks and Mitigation**

| **Risk** | **Mitigation Strategy** |
| --- | --- |
| Security issues | Implement password encryption |
| Data loss | Regular backups for file/database |
| Performance lag | Optimize data structures and transactions |

**6. Non-Functional Requirements:**

* **Security:** Password encryption, authentication validation.
* **Performance:** Fast execution of banking transactions.
* **Scalability:** Handle multiple users and transactions.
* **Usability:** User-friendly interface with clear instructions.

**7. Flow chart**

****

**8. Code**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_USERS 100

// Structure to store user details

typedef struct {

char username[50];

char password[50];

char mobile[15];

char email[50];

int account\_number;

float balance;

} User;

User users[MAX\_USERS];

int user\_count = 0;

// Function prototypes

void createUser();

void loginUser();

void adminLogin();

void transferMoney();

void viewAccountDetails(User \*user);

void adminOperations();

int main() {

int choice;

while (1) {

printf("\nBanking Application\n");

printf("1. User Login\n");

printf("2. Admin Login\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

loginUser();

break;

case 2:

adminLogin();

break;

case 3:

exit(0);

default:

printf("Invalid choice. Try again.\n");

}

}

return 0;

}

// Function to create a new user (Admin only)

void createUser() {

if (user\_count >= MAX\_USERS) {

printf("User limit reached!\n");

return;

}

User newUser;

printf("Enter username: ");

scanf("%s", newUser.username);

printf("Enter password: ");

scanf("%s", newUser.password);

printf("Enter mobile number: ");

scanf("%s", newUser.mobile);

printf("Enter email: ");

scanf("%s", newUser.email);

newUser.account\_number = 1000 + user\_count;

newUser.balance = 0.0;

users[user\_count++] = newUser;

printf("User created successfully! Account Number: %d\n", newUser.account\_number);

}

// Function for user login

void loginUser() {

char username[50], password[50];

printf("Enter username: ");

scanf("%s", username);

printf("Enter password: ");

scanf("%s", password);

for (int i = 0; i < user\_count; i++) {

if (strcmp(users[i].username, username) == 0 && strcmp(users[i].password, password) == 0) {

printf("Login successful!\n");

viewAccountDetails(&users[i]);

return;

}

}

printf("Invalid credentials!\n");

}

// Function to view account details

void viewAccountDetails(User \*user) {

printf("Account Number: %d\n", user->account\_number);

printf("Balance: $%.2f\n", user->balance);

}

// Function for money transfer

void transferMoney() {

int senderAcc, receiverAcc;

float amount;

printf("Enter your account number: ");

scanf("%d", &senderAcc);

printf("Enter recipient's account number: ");

scanf("%d", &receiverAcc);

printf("Enter amount to transfer: ");

scanf("%f", &amount);

User \*sender = NULL, \*receiver = NULL;

for (int i = 0; i < user\_count; i++) {

if (users[i].account\_number == senderAcc) sender = &users[i];

if (users[i].account\_number == receiverAcc) receiver = &users[i];

}

if (sender == NULL || receiver == NULL) {

printf("Invalid account details!\n");

return;

}

if (sender->balance < amount) {

printf("Insufficient funds!\n");

return;

}

sender->balance -= amount;

receiver->balance += amount;

printf("Transfer successful!\n");

}

// Function for admin login

void adminLogin() {

char adminUser[50], adminPass[50];

printf("Enter admin username: ");

scanf("%s", adminUser);

printf("Enter admin password: ");

scanf("%s", adminPass);

if (strcmp(adminUser, "admin") == 0 && strcmp(adminPass, "admin123") == 0) {

printf("Admin login successful!\n");

adminOperations();

} else {

printf("Invalid admin credentials!\n");

}

}

// Admin operations

void adminOperations() {

int choice;

while (1) {

printf("\nAdmin Menu\n");

printf("1. Create User\n");

printf("2. Transfer Money\n");

printf("3. Logout\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

createUser();

break;

case 2:

transferMoney();

break;

case 3:

return;

default:

printf("Invalid choice!\n");

}

}

}