**Project Documentation**

**ATM Project Plan**

Course: Database Systems

Project Overview

The ATM (Automated Teller Machine) project aims to develop a robust and user-friendly system for handling financial transactions. The project will focus on creating software that can interact with hardware components to provide customers with access to banking services such as withdrawing cash, depositing funds, transferring money, and checking account balances.

# Group Formation

1. There are three (3) members in our group
2. The names of the group members
   1. **Mirkomil Mirzohidov**, ID: **221408**, CS6-22
   2. **Muhammad Saidahmetov**, ID: **220838**, CS1-22
   3. **Asilbek Tashpulatov**, ID: **221443**, CS3-22

# Project Details

1. **Requirements Gathering**:

- Conduct interviews and meetings with stakeholders, including bank representatives and potential users, to gather requirements and understand the needs of the ATM system.

- Document functional and non-functional requirements, prioritizing features based on their importance and feasibility.

2. **Design Phase**:

- Design the system architecture, including the interaction between software components and hardware peripherals.

- Create wireframes and mockups for the user interface, focusing on usability and accessibility.

- Define data models and database schemas to store transaction records and customer information securely.

3. **Development**:

- Implement the software components of the ATM system using appropriate programming languages and frameworks.

- Integrate hardware interfaces to communicate with ATM peripherals such as card readers, cash dispensers, and receipt printers.

- Develop algorithms for transaction processing, security validation, and error handling.

4. **Security Implementation**:

- Implement encryption algorithms to secure communication channels between the ATM software and banking servers.

- Integrate PIN verification mechanisms to authenticate users and prevent unauthorized access to accounts.

- Apply security best practices to safeguard against common threats such as card skimming, phishing attacks, and malware infiltration.

5. **Hardware Integration**:

- Test hardware components for compatibility and reliability with the ATM software.

- Configure communication protocols to ensure seamless interaction between software and hardware peripherals.

- Conduct integration testing to verify the functionality of all ATM features, including cash withdrawals, deposits, and balance inquiries.

6. **Testing and Quality Assurance**:

- Develop test cases to validate the functionality, performance, and security of the ATM system.

- Conduct various types of testing, including unit testing, integration testing, system testing, and acceptance testing.

- Perform security assessments and vulnerability scans to identify and mitigate potential risks.

7. **Documentation**:

- Create comprehensive documentation covering system architecture, design decisions, and implementation details.

- Develop user manuals and training materials to guide bank staff and customers on how to use the ATM system effectively.

- Prepare developer guides and troubleshooting resources to aid in system maintenance and support.

8. **Deployment**:

- Coordinate with bank personnel to install and configure ATM hardware at designated locations.

- Deploy the ATM software on server infrastructure, ensuring scalability and reliability.

- Conduct user acceptance testing (UAT) to validate the system's readiness for production use.

9. **Maintenance and Support**:

- Establish procedures for ongoing maintenance, including software updates, patch management, and bug fixes.

- Provide technical support to address any issues or inquiries from bank staff and customers.

- Monitor system performance and security posture, implementing proactive measures to mitigate potential risks and vulnerabilities.

# Database Design

Database Design:

1. **Entities and Attributes**:

- **Customer**:

- Attributes: CustomerID (Primary Key), Name, Address, Phone Number, Email, PIN.

- **Account**:

- Attributes: AccountNumber (Primary Key), CustomerID (Foreign Key), AccountType, Balance.

- **Transaction**:

- Attributes: TransactionID (Primary Key), AccountNumber (Foreign Key), TransactionType, Amount, Timestamp.

2. **Relationships**:

- One-to-Many relationship between Customer and Account (One customer can have multiple accounts).

- One-to-Many relationship between Account and Transaction (One account can have multiple transactions).

3. **Normalization**:

- Ensure data integrity and minimize redundancy through normalization techniques:

- **First Normal Form (1NF):** Each attribute contains only atomic values.

- **Second Normal Form (2NF)**: No partial dependencies; attributes fully depend on the primary key.

- **Third Normal Form (3NF):** No transitive dependencies; attributes depend only on the primary key.

4. **Table Structure**:

- **Customer Table**:

- CustomerID (Primary Key)

- Name

- Address

- Phone Number

- Email

- PIN

- **Account Table**:

- AccountNumber (Primary Key)

- CustomerID (Foreign Key)

- AccountType

- Balance

- **Transaction Table**:

- TransactionID (Primary Key)

- AccountNumber (Foreign Key)

- TransactionType

- Amount

- Timestamp

5. **Indexes**:

- Create indexes on frequently queried columns for faster data retrieval:

- CustomerID in the Customer table.

- AccountNumber in the Account table.

- AccountNumber and Timestamp in the Transaction table.

6. **Constraints**:

- Apply constraints to maintain data integrity:

- Foreign key constraints to enforce referential integrity between related tables.

- Check constraints to enforce business rules (e.g., positive balance, valid transaction types).

7. **Data Types**:

- Use appropriate data types to optimize storage and ensure data accuracy:

- Integer for primary and foreign keys.

- Varchar for variable-length string data.

- Decimal for monetary values to preserve precision.

- Timestamp for recording transaction timestamps.

8. **Normalization**:

- **First Normal Form (1NF)**:

- Split multi-valued attributes into separate columns.

- **Second Normal Form (2NF)**:

- Remove partial dependencies by moving non-key attributes to separate tables.

- **Third Normal Form (3NF):**

- Eliminate transitive dependencies by creating separate tables for attributes that depend on non-key attributes.

9. **Backup and Recovery**:

- Implement regular database backups and recovery procedures to prevent data loss and ensure system availability in case of failures.

10. **Scalability**:

- Design the database schema with scalability in mind to accommodate future growth in data volume and user base.

- Consider partitioning large tables, optimizing queries, and scaling database resources as needed.

4. Tools and Technologies

1. **Programming Languages**:

- Java: Used for backend development, including business logic and database interaction.

- HTML/CSS/JavaScript: Utilized for frontend development to create interactive user interfaces.

- SQL: Employed for database querying and manipulation.

2. **Frameworks and Libraries**:

- Spring Framework: Provides comprehensive support for building enterprise Java applications, including dependency injection and MVC architecture.

- Hibernate: Object-relational mapping (ORM) framework used to simplify database interactions and manage persistence.

- React.js or Angular: Frontend frameworks for building responsive and dynamic user interfaces.

- Bootstrap: Frontend framework for designing mobile-first and responsive web pages.

3. **Database Management System (DBMS)**:

- MySQL or PostgreSQL: Open-source relational database management systems (RDBMS) used for storing and managing transactional data.

4. **Integrated Development Environment (IDE)**:

- IntelliJ IDEA or Eclipse: Feature-rich IDEs for Java development, offering tools for code editing, debugging, and version control integration.

- Visual Studio Code: Lightweight IDE for web development with support for various programming languages and extensions.

5. **Version Control**:

- Git: Distributed version control system for tracking changes in source code and collaborating with development teams.

- GitHub or GitLab: Platforms for hosting Git repositories, facilitating code sharing, collaboration, and project management.

6. **Continuous Integration and Deployment (CI/CD)**:

- Jenkins: Automation server for orchestrating CI/CD pipelines, including building, testing, and deploying software.

- Docker: Containerization platform for packaging applications and their dependencies into lightweight containers, ensuring consistency across different environments.

- Kubernetes: Container orchestration platform for automating deployment, scaling, and management of containerized applications.

7. **Security Tools**:

- OWASP ZAP: Open-source security testing tool for identifying vulnerabilities in web applications.

- Burp Suite: Integrated platform for performing security testing, including scanning for web vulnerabilities and testing API security.

- Keycloak: Open-source identity and access management solution for securing applications with features like single sign-on and multi-factor authentication.

8. **Monitoring and Logging**:

- ELK Stack (Elasticsearch, Logstash, Kibana): Stack for centralized logging and real-time log analysis, providing insights into application performance and system behavior.

- Prometheus: Monitoring and alerting toolkit for collecting and querying metrics from distributed systems.

- Grafana: Open-source analytics and visualization platform for monitoring and visualizing metrics from multiple data sources.

9. **Communication and Collaboration**:

- Slack or Microsoft Teams: Messaging and collaboration platforms for team communication, file sharing, and project coordination.

- Jira or Trello: Project management tools for planning, tracking, and managing software development tasks and workflows.

5. Submission Instructions

1. **Submission Format**:

- Please submit your project in a digital format, preferably as a compressed file (e.g., ZIP or RAR) containing all necessary files and documentation.

- Ensure that the file is named appropriately, indicating the project name or code, along with your name or team identifier if applicable.

2. **Documentation**:

- Include comprehensive documentation covering all aspects of the project, including design decisions, implementation details, and usage instructions.

- Provide user manuals and guides to assist users in understanding and utilizing the system effectively.

- Include any relevant diagrams, such as system architecture, database schema, and user interface wireframes.

3. **Source Code**:

- Include all source code files necessary for compiling and running the project.

- Organize the codebase into logical directories and provide clear instructions for building and running the application.

4. **Database Scripts**:

- Include scripts for creating the database schema, tables, and any necessary seed data.

- Ensure that database scripts are compatible with the chosen database management system (e.g., MySQL, PostgreSQL).

5. **Testing and Quality Assurance**:

- Provide documentation of testing procedures and results, including test plans, test cases, and test reports.

- Include any automated testing scripts or tools used during the development process.

6. **Deployment Instructions**:

- Include detailed instructions for deploying the application in a development or production environment.

- Specify any system requirements, dependencies, and configuration settings necessary for successful deployment.

7. **Additional Resources**:

- If applicable, include any additional resources such as design mockups, prototypes, or demo videos to enhance understanding of the project.

8. **Submission Method**:

- Submit your project via the designated platform or email address provided by the instructor or project coordinator.

- Ensure that the submission is made within the specified deadline and follows any additional guidelines or requirements provided.

9. **Contact Information**:

- Include your contact information (e.g., email address, phone number) in case there are any questions or clarifications needed regarding your submission.

10. **Acknowledgement of Receipt**:

- If required, request acknowledgment of receipt from the recipient upon submission to ensure that your submission has been received and logged.

# Summary

**Project Summary:**

The ATM (Automated Teller Machine) project aims to develop a robust and user-friendly system for handling financial transactions. The project encompasses various stages, including requirements gathering, design, development, testing, deployment, and maintenance.

Key objectives of the project include designing an intuitive user interface, implementing core functionalities such as cash withdrawal and balance inquiry, integrating robust security measures, and ensuring seamless hardware integration.

The database design features entities such as customers, accounts, and transactions, with relationships established to maintain data integrity. Normalization techniques are applied to minimize redundancy and ensure efficient data storage.

The project utilizes a range of tools and technologies, including Java for backend development, HTML/CSS/JavaScript for frontend design, MySQL or PostgreSQL for database management, and frameworks such as Spring and Hibernate for application development.

Submission instructions outline the required format for submitting the project, including documentation, source code, database scripts, testing procedures, deployment instructions, and additional resources. Submissions are made via the designated platform or email address within the specified deadline.

Overall, the ATM project aims to deliver a comprehensive and reliable system that meets the needs of stakeholders while adhering to industry standards and best practices in software development.