Report

# Section 1: Description and Evidence of Agile and XP Techniques

## Project Overview:

It is a banking application that has been developed using Next.js and MongoDB providing functionality for account management, financial transactions, account services as well as security and compliance. The application provides a very simple user interface with powerful support at the back to be able to deal with many operations that take place in the banking system.

## Agile Methods Employed:

### Iterative Development:

* **Approach:** The application was developed by breaking down the project into small, manageable iterations (sprints). Every sprint aimed to deliver a subset of features or functionality with continuous feedback and improvement. This way, from each sprint output is obtained, which is demonstrated to obtain confirmation toward the end of it.
* **Evidence:** Features, such as account management and financial transactions, were developed in independent sprints. For example, if the first sprint focused on the creation and setup of accounts, then the second sprint would be about financial transactions like deposits or withdrawals.

### User Stories and Backlog Management:

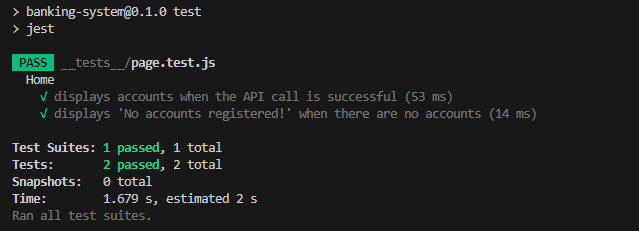
* **Approach:** The project captured requirements from an end-user perspective. Evidence, each user story was added to the product Backlog, prioritized, and refined in collaboration with stakeholders.
* **Evidence:** It had such user stories as "As a user I want to be able to deposit cash and have the ATM give me a receipt" created and added to the backlog. The backlog is reviewed regularly and adjusted due to feedback and changing priorities. See appendix for user stories.

### Sprint Planning and Review:

* **Approach:** The sprint planning meetings were done at the beginning of each sprint to determine the scope and goals. The sprint review meetings were held at the end of each sprint to display the completed features and get feedback.
* **Evidence:** When using the sprint planning, implementation of some desirable items like the possibility to implement deposit functionality and the generation of receipts were defined and assigned. Reviews of the sprint work were achieved by presenting what has been developed: the working deposit feature – and then discussing possible enhancements.

### Continuous Integration and Deployment:

* **Methodology**: To ensure that code updates were incorporated into the primary project consistently, the project had git version control. In order to allow for smooth releases, Jest. js was utilized for scripting of automated tests probably since it can be.



### Pair Programming and Code Reviews:

* **Approach:** That is why such practices as pair programming and code reviews, which enhance the quality and maintain teamwork among participants, were promoted among participants.
* **Evidence:** The concept of the pair programming was applied for such important areas of the code, for instance, two-factor mechanisms of the authentication. In order to ensure that the best practices and coding standards were implemented, there were code reviews done.

### Test-Driven Development (TDD):

* **Method:** Prior to unleashing these features, writing of tests was done using test-driven development. As for this method it helped in being able to state the requirements clearly and achieving oversight to ensure that the code functioned in the right manner.
* **Evidence:** Before the deployment, tests for the aspects such as fund-transferring and the transaction history were set a bit. It ensured that all the features were fully operational, and had been thoroughly checked by the testers.

## Hypothetical Group Development:

### Roles and Responsibilities:

* **Product Owner:** Understood the goals of the project, prioritization of the backlog, and ensured that the development team delivered components that met the goal of the business.
* **Scrum Master:** He was in charge of leading stand-ups, retrospectives, as well as sprint planning. minimised the barriers and at the same time checked that they were following Agile processes.
* **Developers:** Were writing code, doing code reviews, and getting feature implementations done. Account management interfaces and back end processing services were some of the features that developers considered in the application.
* **Testers for quality assurance (QA):** The application was reviewed in order to check for mistakes and ensure that those features aligned to the criteria for approval. They never set up dummy runs regarding the features they offered to their customers; rather, they developed test scenarios for the assessment of their own features.

### Dividing Responsibilities:

* Account creation, editing, and termination were the responsibility of a single team member.
* Financial Transactions: The deposit, withdrawal, and fund transfer processes were managed by a different team member.
* Account Services: A third team member handled checkbooks, regular payments, and account information.
* Security and Compliance: A fourth team member made sure that data encryption and strong authentication procedures were in place.

### Planning:

* Sprint Planning: During the sprint planning sessions, the team was able to identify the tasks required, the individual, team, or sub-team that was to be responsible for it, as well as an estimation by the team of the effort required to complete this particular task.
* Daily Stand-Ups: Daily huddles were made to discuss the day’s plan and the challenges faced and accomplishments in the day.
* Sprint Retrospectives: At the end of each sprint the team conducted a short-get-together or a post-analysis to point out the strength of the project and the weakness that was observed and on areas that required change or enhancement.
* It was an integrated project that was developed in an incremental way using XP and Agile methodologies that allowed flexibility in the face of changing requisites.

# Section 2: Project Planning

## Software Process Selection:

Out of all the methodologies for banking application development agile software development was chosen because of its nature in relation to several iterations and thus the fact that it is a strong methodology for a project which goes through a lot of change. It is due to this reason that agile methods help in managing complex project of dynamic requirements by providing for continuous feedback, improvement and change.

## Reasoning for Choosing Agile:

### Flexibility and Adaptability:

* **Reasoning:** The situation in banking programs frequently requires changes due to the changes in the regulations, the feedback received from users, and the changes in the technology. Finally, agile makes the working on the application up-to-date and relevant because it allows for numerous modifications.
* **Evidence:** This is advantageous because, for instance, if there is a shift of user requirements, or a shift in the legal requirements during the course of the development phase, these can be captured in the subsequent sprints.

## User-Centric Approach:

* **Reasoning:** Pretty much everything one does in Agile; it is all about providing users with the act of value. The development team can utilize the features that will in one way or the other will solve the user’s need and improve their experience all together through interacting with the stakeholders and even the users themselves.
* **Proof:** Some of the functional user stories included ‘‘View detailed Account information’’ which was created to meet specific needs and was prioritized according to levels of satisfaction of the user.

## Continuous Feedback and Improvement:

Agile adopts iterative development and regular feedback, which assist in improving features and resolving problems early in the development cycle. Moreover, this means that the application evolves based on feedback from stakeholders and real-world usage.

Basing of my evidence on two meetings, I would like to state that sprint reviews provided opportunities for input while retrospectives facilitated evaluations of developments to make necessary changes where needed.

## Breaking Down the Work into Activities and Milestones:

A screenshot of a calendar

Description automatically generated

### Initial Planning:

* **Activities:**
  + Define project scope and objective.
  + Identify key featuers and functionalities.
  + Create a high level project roadmap.
* **Milestones:**
  + Project kickoff and initial planning completed.
  + High-level requirement and goals established.

### Requirement Analysis and Backlog Creation:

* **Activities:**
  + Gather detailed requirement from stakeholders.
  + Break down requirement into user stories and add them to the product backlog.
  + Prioritize backlog item based on importance and dependency.
* **Milestones:**
  + Product backlog created and prioritized.
  + Detailed requirement documented and reviewed.

### Sprint Planning and Execution:

* **Activities:**
  + Plan sprints, including defining sprint goals and selecting backlog item for implementation.
  + Develop features based on sprint goal, including coding, testing, and integrating functionalities.
  + Conduct daily stand-up meetins to track progress and address issues.
* **Milestones:**
  + Completion of each sprint with defined features and functionalities.
  + Successful integration and testing of developed features.

### Feature Development and Integration:

* **Activities:**
  + Implement core feature such as account management, financial transactions, and account services.
  + Integrate security and compliance measure into the application.
  + Conduct code review and ensure adherence to coding standards.
* **Milestones:**
  + Core features developed and integrateds.
  + Security and compliance measures implemented.

### Testing and Quality Assurance:

* **Activities:**
  + Conduct unit testing, integration testing, and system testing.
  + Perform user acceptance testings (UAT) with stakeholders.
  + Address bugs and issues identified during testing.
* **Milestones:**
  + Successful completion of testing phases.
  + Bug fixes and improvements based on testing feedback.

### Deployment and Release:

* **Activities:**
  + Prepare the application for deployment to production environments.
  + Conduct final checks and validation before release.
  + Deploy the application and monitor for post-release issues.
* **Milestones:**
  + Application deployed to production.
  + Post-release monitoring and support initiated.

### Post-Deployment and Maintenance:

* **Activities:**
  + Collect user feedback and monitor application performance.
  + Address any issues or bugs reported by users.
  + Plan and implement updates and enhancements based on feedback.
* **Milestones:**
  + Post-deployment feedback collected and analyzed.
  + Updates and enhancements planned for future sprints.

## Summary:

The Agile methodology was chosen because of its adaptability to changing demands and emphasis on creating value through incremental development. The project was separated into discrete tasks with completion dates to ensure structured development and optimal resource utilization. With clearly defined goals and deliverables for each step of the project, the team was able to stay on track and achieve the desired results.

# Section 3: Prototype Design

## System and Interface Design Overview:

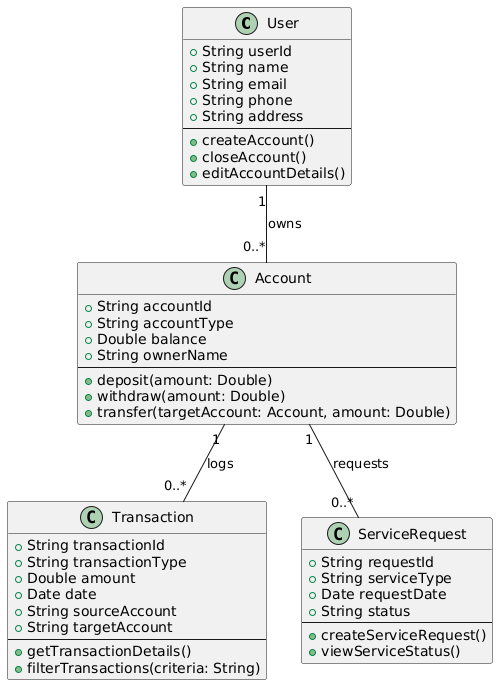
The banking application prototype was created to provide an easy-to-use interface for conducting banking transactions. The design includes the overall architecture of the system, as well as the user interface components and interactions. This section provides a detailed discussion of the interface and system designs, as well as screenshots of the completed prototype and automated testing.

## System Design:

The system design is represented by class diagrams, sequence diagrams, and use case diagrams, which show the application's structure and behavior.

### Class Diagram:

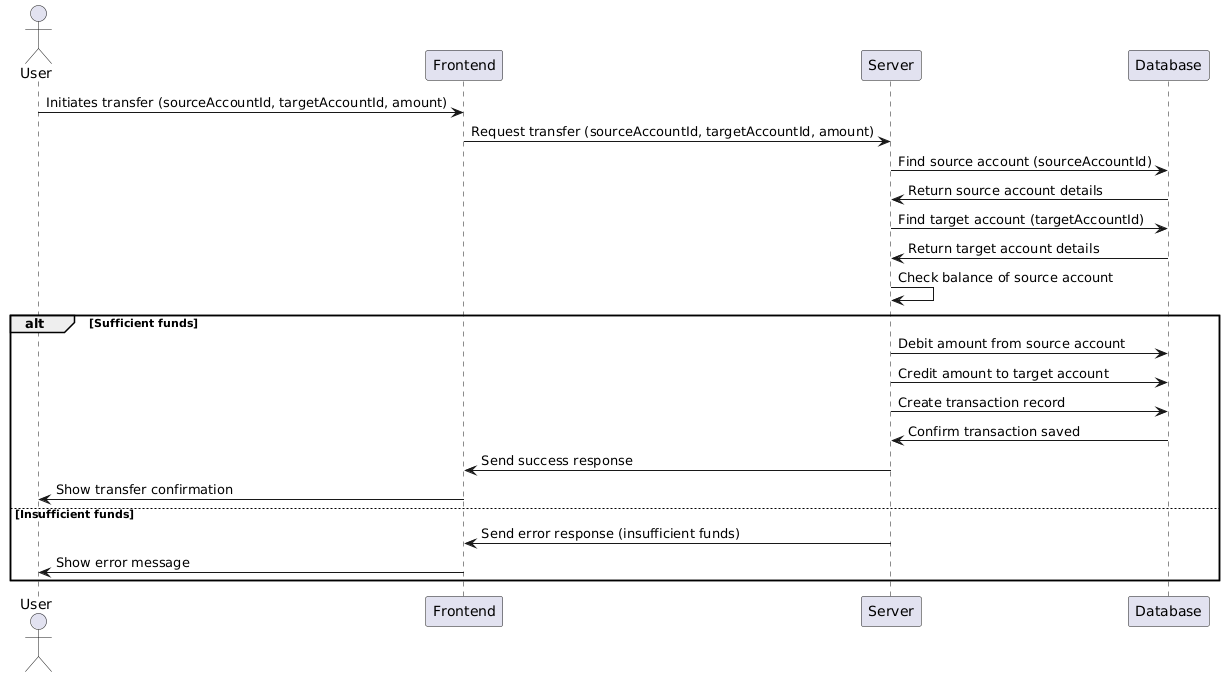
* **Description:** The class diagram depicts the major pieces of the banking application and their connections. They include classes for managing transactions, account services, and user accounts.



* **Details:**
  + Account class manages account details and operations.
  + Transactions class handle financial transactions.
  + User class represents users of the application and their profiles.
  + Service class manages additional account services like checkbooks and cards.

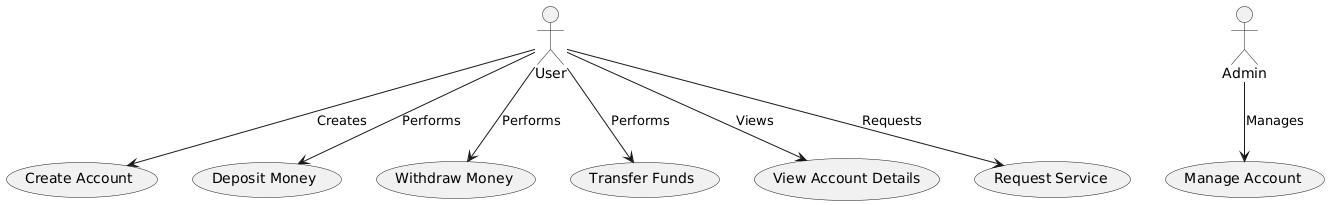
### Sequence Diagram:

* **Description:** Sequence diagrams show how various system components interact with one another during particular use cases, including financial transfers or account creation.



### Use Case Diagrams:

* **Description:** Use case diagrams outline the system's functionality from the viewpoint of the user as well as the various actors' interactions with it.

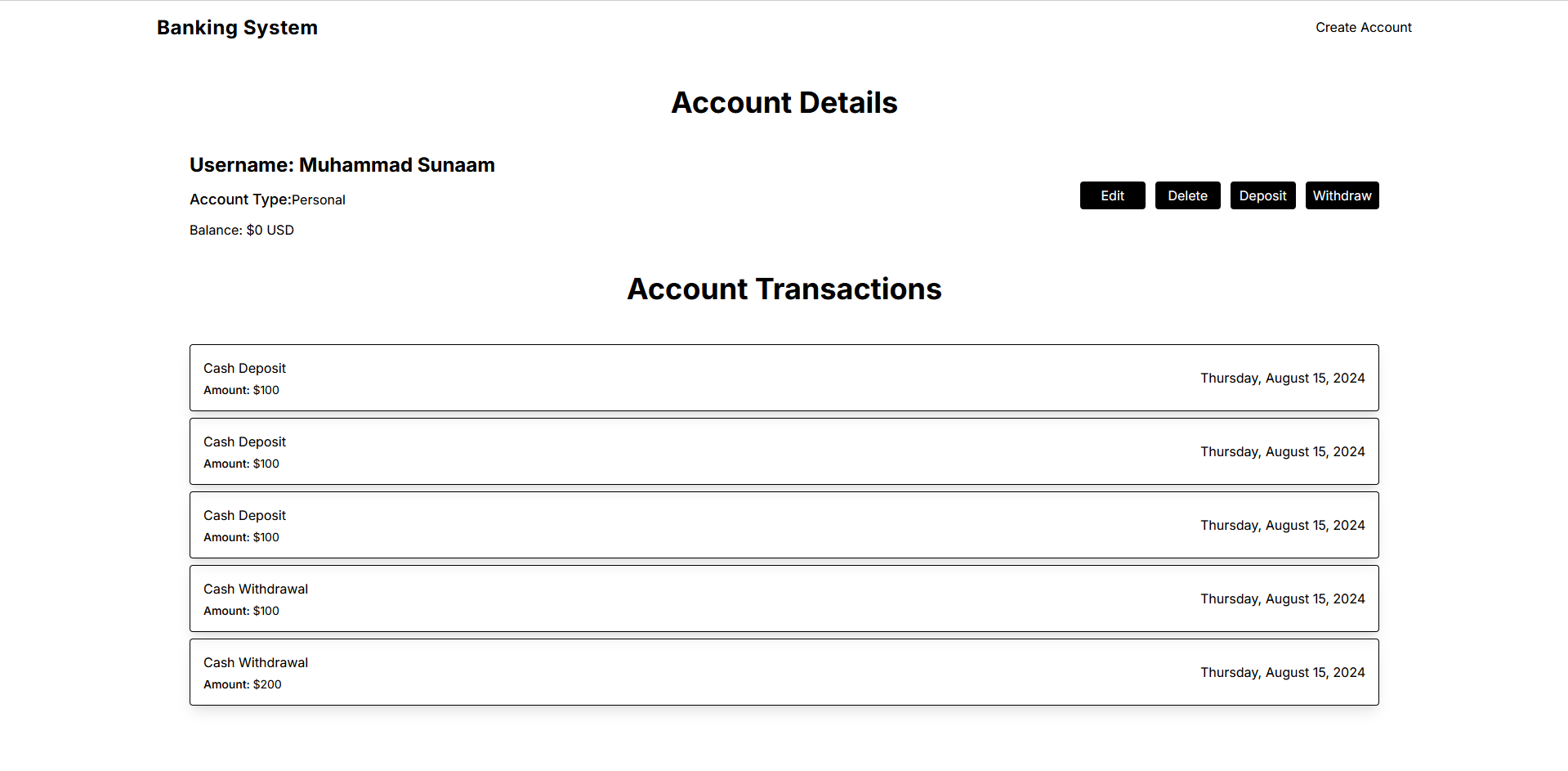
****

* **Details:**
  + Bank Customer can perform operations like account management, financial transactions, and requesting services.
  + System Administrator handles administrative tasks such as managing user accounts and ensuring system compliance.

## Interface Design:

### User Interface (UI) Design:

* **Description:** The goal of the user interface design is to make the banking application's interface simple and easy for people to use. Transaction forms and dashboards are important components.
* **Screens:**
  + **Dashboard:** Displays a summary of account information, recent transactions, and quick access to key features.



* + **Transaction Forms:** Simple and user-friendly forms for depositing, withdrawing, and transferring funds.

A screenshot of a computer

Description automatically generated

### Evidence of Developed Prototype:

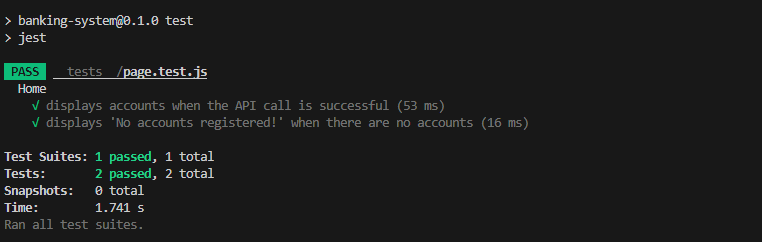
Link:

## Automated Testing:

### Automated Testing Overview:

Automated testing was implemented to ensure the banking application's reliability and functionality. The test designs incorporated aspects like functional integration and security testing.

* **Instrument Applied:** The tests were written and run using Jest as the testing framework, guaranteeing thorough coverage and trustworthy outcomes.



# Section 4: Critical Evaluation of Software Engineering Tools and Techniques

## Advances in Software Processes

### Methodologies:

To ensure an orderly and productive development process, the banking application project utilized modern software development methodologies. Agile (Torgeir, Sridhar, VenuGopal, & Nils Brede, 2012) was the primary technique used, and its flexible and iterative nature proved effective.  
By utilizing Agile approach, the project was able to quickly incorporate feedback and adapt to changing demands. For example, feedback on the first prototype increased the usability of the account management interface. Agile's iterative cycles enabled incremental development and frequent re-evaluation of priorities, which was critical for handling the banking application's dynamic requirements.

### Agile Processes:

The development of the banking application was properly managed using agile techniques. The project uses continuous integration, sprint planning (Inayat, Salim, Marczak, Daneva, & Shamshirband, 2015), and iterative development methodologies to improve the development environment's flexibility and reactivity.  
Sprints were used to create and polish elements such as account services and financial transactions, allowing for regular review and adjustment. Sprint retrospectives and daily stand-up meetings provided opportunities for continuous improvement and problem solving. For example, during a sprint review, the team determined that additional security features were required, and those capabilities were swiftly added to the next sprint cycle.

## Software Engineering Techniques

### Project Management:

Effective project management was required to ensure the successful development of the banking application. Backlog management, sprint planning, progress monitoring, and other agile project management methods (Highsmith) were utilized to manage tasks and ensure feature delivery on time.  
The product backlog was handled, and the progress of each sprint was tracked with Jira and other similar technologies. Sprint goals were established, and tasks were ranked based on their dependencies and importance. This strategy ensured that critical features, such as transaction processing and secure login, were completed and tested on time.

### Prototype Design:

According to Rogers, Sharp, and Preece (2011), the prototype design phase involved creating user-friendly interfaces and validating them through iterative feedback. Class, sequence, and use case diagrams enabled a full understanding of the system architecture and user interactions.  
The interface design included mockups and screenshots of key components such as the account dashboard and transaction forms. Prototype testing provided feedback that helped refine the design, resulting in a more user-friendly and intuitive experience. In response to user feedback, the layout of the transaction history page was changed to improve readability.

### Version Control:

Git and GitHub were utilized to manage version control, facilitating communication, code management, and change tracking throughout the development process.  
Git was used for branching, merging changes, and managing code versions. GitHub provided a framework for collaborative development, problem tracking, and code reviews. On GitHub, pull requests and commit history demonstrate how code changes were implemented and analyzed, ensuring code coherence and quality.

## Impact of Advanced Software Systems and Engineering

### Social Impact:

Innovative digital technologies have transformed how customers interact with banking services, improving the efficiency and accessibility of financial management. The banking application allows users to effortlessly execute transactions, view account information, and manage their finances online.  
The program improves accessibility and user experience by providing a user-friendly interface and features like as real-time balance updates and transaction history. The convenience of banking from a mobile device has increased, and people are more prepared to manage their money.

### Ethical Impact:

The banking application was designed with ethical considerations in mind, particularly in terms of data protection and privacy. To maintain user confidence and protect data, make sure that sensitive data is protected and that the application complies with applicable regulations.  
To protect user information, the application encrypts important data and employs robust authentication mechanisms like two-factor authentication. To address ethical data security challenges, compliance with financial standards and data protection legislation was given primary emphasis.

### Entrepreneurial Impact:

Banking apps are an illustration of the fintech industry's potential for commercial endeavors. The project highlights how entrepreneurs can develop and give new solutions in the financial industry by leveraging modern technologies and software engineering methods.  
The application's construction emphasizes the possibility of creating user-centered, competitive financial services. With features such as safe account management and automated transactions, the app is positioned as a valuable resource in the fintech industry, offering opportunities for growth and self-employment.

# References

Highsmith, J. A. (n.d.). *Adaptive Software development: a collaborative approach to managing complex systems.*

Inayat, Salim, Marczak, Daneva, & Shamshirband. (2015). A systematic literature review on agile requirements engineering practices and challenges. *Computers in Human Behavior*.

Rogers, Sharp, & Preece. (2011). *Interaction Design: Beyond Human-Computer Interaction.*

Torgeir, Sridhar, VenuGopal, & Nils Brede. (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*.

# Appendix

**Video Demonstration:** [Link to video demonstration of the prototype]

**Repository:** <https://github.com/AyeshaSaddiqa001/BankingSystem_1>

**User Stories:** see stories.md on Github.