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:

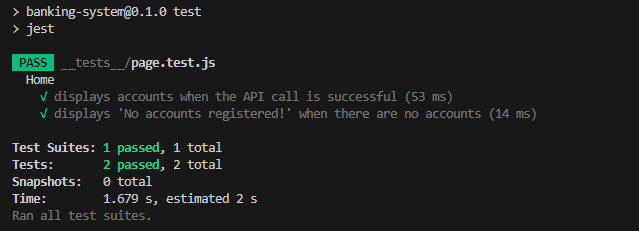
* **Approach:** The project captured requirements from an end-user perspective. Evidence, each user story was added to the product Backlog, prioritized, and refined in collaboration with stakeholders.
* **Evidence:** It had such user stories as "As a user I want to be able to deposit cash and have the 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:**
  + We implemented core features including accounts management, financial transaction management, and service management.
  + We integrated security features into the application.
  + We conducted code reviews to ensure coding standards.

### 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 selected due to its flexibility in responding to changing requirements and its focus on iterative development. The project was divided into distinct tasks, each with set completion dates, to facilitate organized development and efficient use of resources. By establishing clear objectives and deliverables for every phase of the project, the team managed to remain focused and attain the expected outcomes.

# Section 3: Prototype Design

## System and Interface Design Overview:

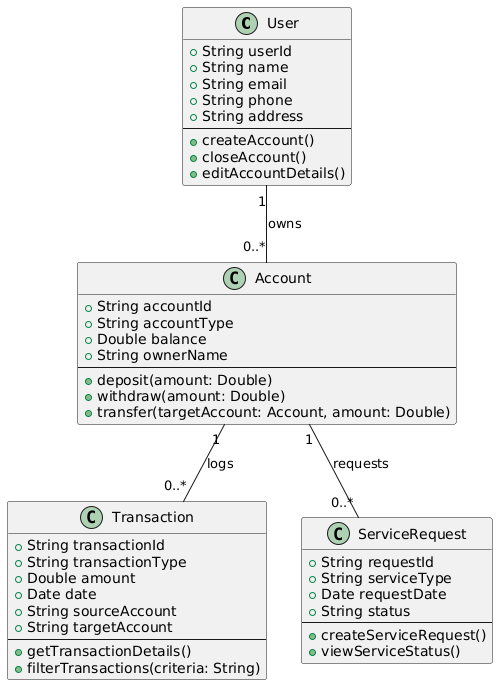
The prototype for the banking application was developed to offer a user-friendly interface for executing banking transactions. The design encompasses both the system's overall architecture and the various user interface elements. In this section, we present an in-depth analysis of the interface and system designs.

## System Design:

Class diagrams are used to present the system design as follows:

### Class Diagram:

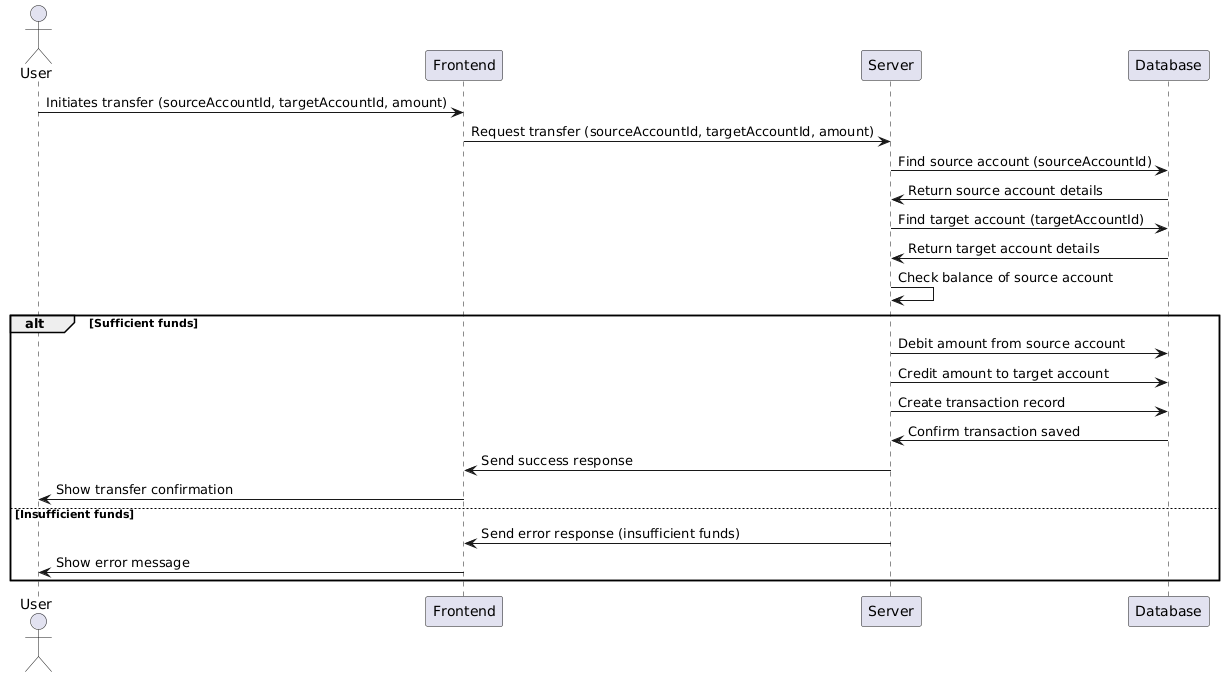
* **Description:** The class diagram shows the major components of the banking application and their connections. They include classes for managing transactions, account services, and user accounts.



* **Details:**
  + Account class will manage user details.
  + Transactions class will handle withdraw, deposit, etc.
  + User class represents users of the application and their profiles.
  + Service class manages additional account services like checkbooks and cards.

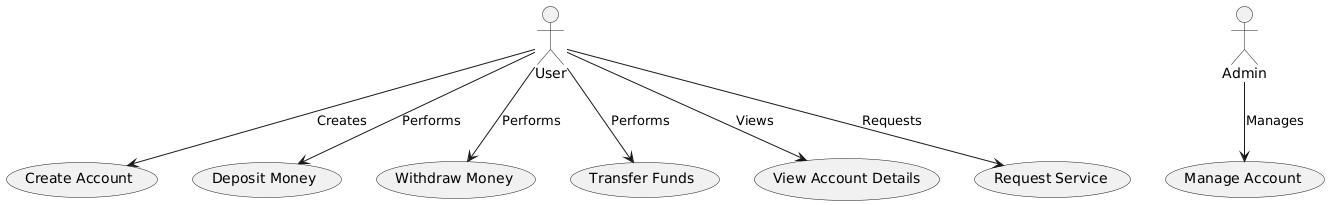
### Sequence Diagram:

* **Description:** Sequence diagrams show how various system components interact with one another during particular use cases.



### 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.

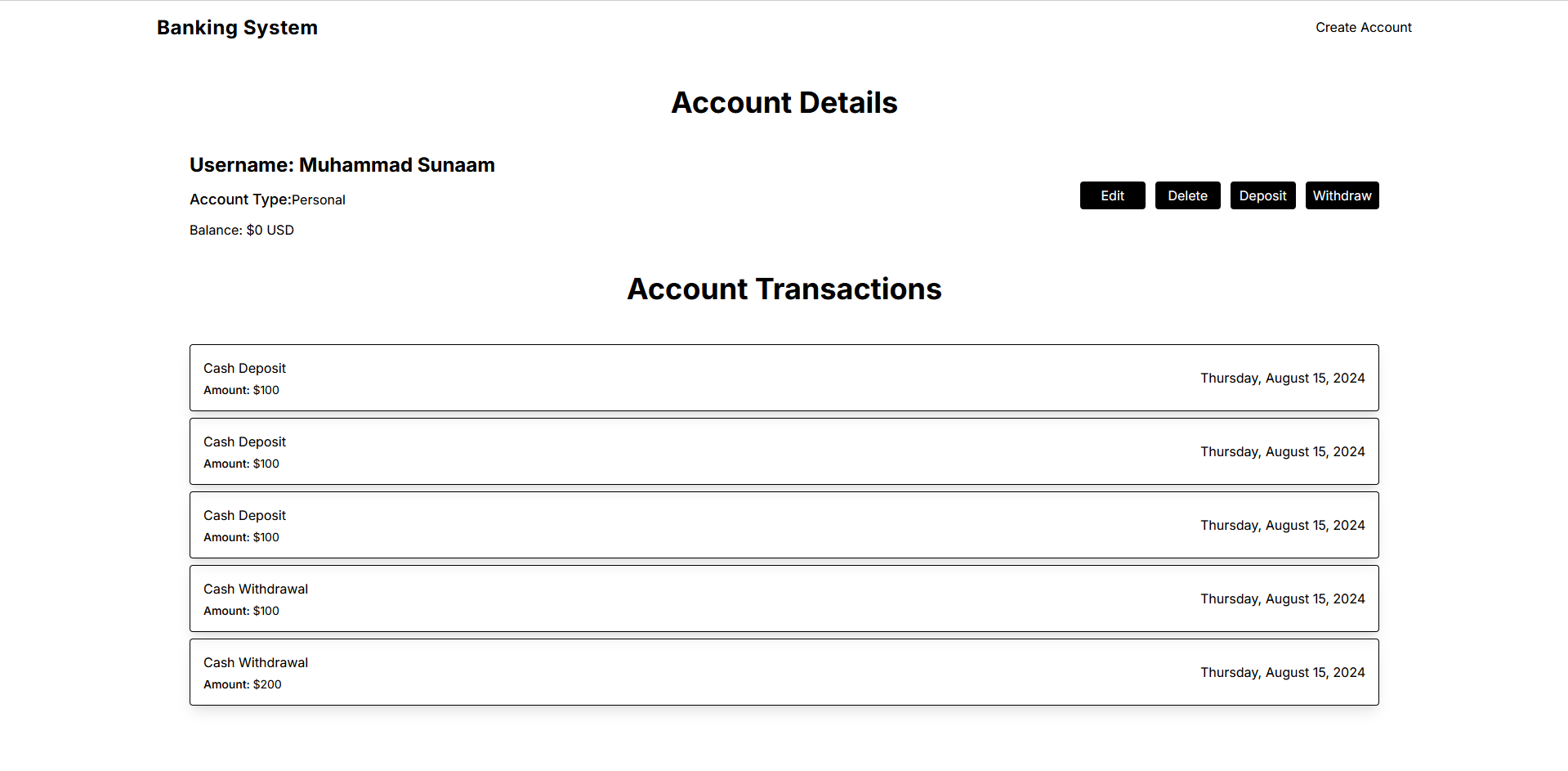
****

* **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

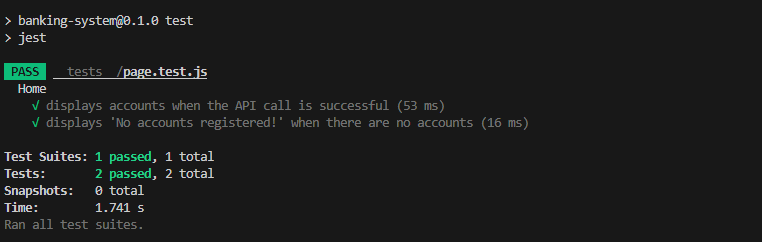
Description automatically generated

## 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 maintain an organized and efficient development process the banking application project employed contemporary software development methodologie. The primary method utilized was Agile (Torgeir, Sridhar, VenuGopal, & Nils Brede, 2012), known for its adaptable and iterative characteristics, which proved to be highly effective.

Leveraging the Agile approach allowed the project to swiftly integrate feedback and adjust to evolving needs. For instance, enhancements based on initial prototype feedback improved the usability of the account management interface. The iterative cycles integral to Agile facilitated gradual development and regular reassessment of priorities, which was essential for accommodating the banking application's ever-changing requirements.

### Agile Processes:

The banking application was developed with effective management through the implementation of agile techniques. The project leverages continuous integration, sprint planning (Inayat, Salim, Marczak, Daneva, & Shamshirband, 2015), and iterative development methodologies to enhance the flexibility and responsiveness of the development environment.

Sprints were utilized to create and refine features such as account services and financial transactions, facilitating regular evaluations and adjustments. Daily stand-up meetings and sprint retrospectives provided avenues for ongoing improvement and problem-solving. For instance, during a sprint review, the team identified a need for enhanced security features, which were promptly incorporated into the upcoming sprint cycle.

## Software Engineering Techniques

### Project Management:

Successful development of the banking application necessitated efficient project management. Agile project management techniques (Highsmith) such as backlog management, sprint planning, and progress tracking were employed to oversee tasks and guarantee timely feature delivery.

The product backlog was meticulously managed, and the advancement of each sprint was monitored using Jira and comparable tools. Specific sprint objectives were set, with tasks prioritized according to their significance and dependencies. This approach ensured that key features, including transaction processing and secure login, were finalized and thoroughly tested within the designated timeline.

### Prototype Design:

According to Rogers, Sharp, and Preece (2011), the prototype design phase emphasized creating user-friendly interfaces and validating them through a process of iterative feedback. The use of class, sequence, and use case diagrams helped achieve a thorough understanding of the system’s architecture and user interactions. The interface design included mockups and screenshots of the key components, such as the account dashboard and transaction forms. Based on user recommendations, adjustments were made to the design of the transaction history page to improve readability.

### Version Control:

Git versioning and GitHub were utilized for managing version control of code organization and tracking changes throughout the development process. GitHub provided a framework for collaborative development, issue tracking, and code reviews. On GitHub, pull requests and the commit history demonstrate how code changes were made and reviewed, ensuring both consistency and quality in the code.

## Impact of Advanced Software Systems and Engineering

### Social Impact:

The banking app allows users to do their transactions as well as check and manage their money online with ease.

Thanks to its intuitive interface and real-time balance updating plus transaction history, the app really does make all the difference in terms of accessibility and user at hand. As mobile banking grows more seamless, it encourages a person to take control over their financial management.

### Ethical Impact:

The banking app was built with a heavy focus on ethical considerations, especially data security and user privacy. Secure handling of information and legislation are essential to building confidence for users seeking to protect their sensitive data.

The application secures user data by encrypting vital information and implements robust authentication mechanisms like 2FA. The ethical issues in data security were related to honoring financial regulations and laws on data protection that in it-self was a commitment.

# 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.