FinEasy: Simplifying Personal Finance

Boyuan YE[1], Jinming ZHANG[2], Qianru CUI[2]

[1]Software Engineering Systems, Northeastern University
[2]Information Systems, Northeastern University
{ye.boy, zhang.jinmin, cui.qianr}@northeastern.edu

Abstract—**In this report, we discussed the development and value of a comprehensive JavaFX application called FinEasy that designed to enhance personal financial management through effective tracking and budgeting. This tool allows users to meticulously log their daily income and expenses, categorize entries, and modify record details or their sequence as needed. It utilizes multiple abstract data types, including Lists, Stacks, Sorting and Binary Search Trees, to manage transactions, facilitate operation recall, and enable quick record searches. These features combine to offer a user-friendly experience that promotes financial organization and accessibility.**

Index Terms—**Abstract Data Type, Stack, List, Binary Search Tree, Sorting, JavaFX application, Financial Management**

I. PROBLEM DESCRIPTION

Personal financial management is a fundamental activity for individuals to achieve financial goals and maintain economic stability [1]. FinEasy, a desktop application developed in JavaFX, is designed to aid users in managing their personal finances by providing tools for recording, categorizing, and visualizing financial transactions. The necessity for such tools stems from the increasing complexity of personal financial landscapes. The burgeoning variety of financial products and the ubiquitous nature of economic transactions demand a systematic approach to financial management. FinEasy aimed at users who need to monitor their financial situation or tracking their expense in a timely manner, especially the young, simplifying the complexity by offering an intuitive platform for users to monitor and analyze their spending and income patterns.

JavaFX [2] was chosen as the development framework for its rich features supporting desktop applications. Its ability to create responsive and aesthetically pleasing user interfaces aligns with the objective of making financial management an engaging experience rather than a taxing chore.

Abstract Data Types (ADTs) are employed within FinEasy to encapsulate the data and operations of financial transactions, which include Stacks, Lists, Sorting, and Binary Search Trees. The choice of ADTs is motivated by their potential to structure data in a way that enables efficient access and manipulation—essential qualities for a tool that handles potentially large volumes of financial data [3].

II. ANALYSIS (RELATED WORK)

With the rise of next generation mobile networks, smartphones, and payment systems have been moving to new and different kinds of mobile platforms and possibilities. As Maurer has shown, mobile payment platforms are social computing systems that support new forms of sociality [4]. The digital payment has made people a extremely convenient way to spend money which have great potential to change the lifestyles of millions of people in developing countries [5]. However, while it brings convenience, it also brings some new problems. According to the research by Ramadani [6], due to the ease and speed of e-payment, the higher the use of digital payment (e-payment), the higher the consumer behaviour of students. Baumeister's research also proved that the failure of keeping track of (monitor) one's own behavior renders control difficult [7].

To address this problem, we decided to introduce our application - FinEasy, offer a user-friendly tool for individuals to monitor and analyze their financial activities, empowering them with the knowledge and insights needed to make informed financial decisions which can help to reduce the temptation of spend money.

III. SYSTEM DESIGN

We used one Chinese online prototype design platform - xiaopiu (Available: https://www.xiaopiu.com/ Accessed on: Apr. 9, 2024.) for the draft user interface design shown in Figure 1. It help us to build the overview of the project and reduced the modification after implementation.
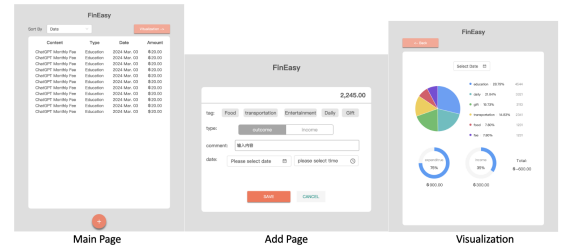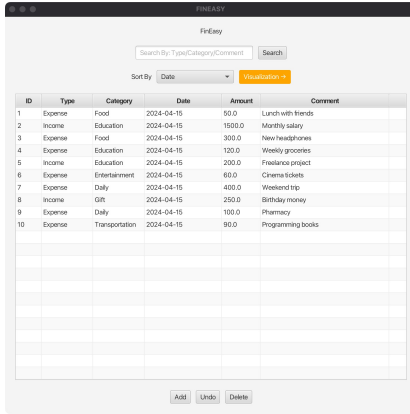

Fig. 1. Prototype

A. Main Page

Figure 2 illustrates the primary interface of the application upon initiation. This interface encompasses a text field that enables users to execute queries using keywords related to type, category, or descriptive comments to retrieve one or several pertinent records.

Central to this interface is a comprehensive transaction table delineating the user's financial activities in meticulous detail. By default, the transactions are arranged in a chronological sequence, with additional functionalities permitting users to reorganize the data based on temporal markers, monetary values, categories, or transaction types.

The interface also integrates a quartet of interactive elements. Prominently positioned at the upper section of the page is the 'Visualization' button; activating this feature redirects to a subsidiary page that graphically delineates the user's financial outlays and income through a dynamic pie chart. Complementary to this, situated at the base of the interface are three distinct buttons—'Add', 'Undo', and 'Delete'. These buttons facilitate the navigation to the pop-up transaction addition interface, the reversion of the most recent action, and the excision of selected transactional entries, respectively.

Fig. 2. Main Page

B. Add Page

Figure 3 depicts the pop-up interface designed for the insertion of new transaction records by the user. The interface is systematically arranged to include an input field for the transaction amount, which is constrained to accept only floating-point or integer data types. Accompanying this field is a selection box delineated for the categorization of the transaction, alongside a drop-down selector engineered to facilitate the specification of the transaction's type. Additional provision is made for the inclusion of ancillary remarks via a designated comment text field, and a date picker tool is incorporated to record the transaction's temporal data accurately.

Upon the completion of data entry, the user may opt to commit the transaction record to the system by engaging the 'Save' button, which consequently registers the transaction and navigates the user back to the main page. Alternatively,

should the user decide against recording the transaction, the 'Cancel' button serves to disregard any input and also returns the user to the main interface without committing any new data to the system record.
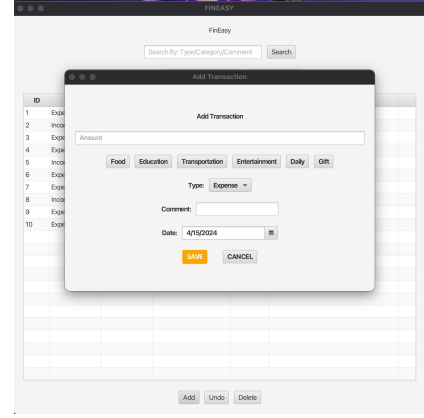

Fig. 3. Transaction Addition Page

C. Visualization Page

The interface shows in Figure 4 displays a financial overview using a multi-colored, which serves as a graphical summary of the user's financial transactions. Each segment of the pie chart is color-coded to correspond to a specific category such as Transportation, Food, Gifts, and more, with the legend provided at the bottom of the chart facilitating quick identification by matching color to category. When mouse hover on each part of the pie chart, the corresponding slice will be highlighted and details of the category will show. And if there' no record for this type (expenditure or income), the corresponding pie chart will disappear.

Key financial figures are summarized below the chart, with clear indications of expenses, income, and net total. A 'Back' button allows for easy navigation, and the application's title. The design prioritizes a clean aesthetic and user-friendly experience.
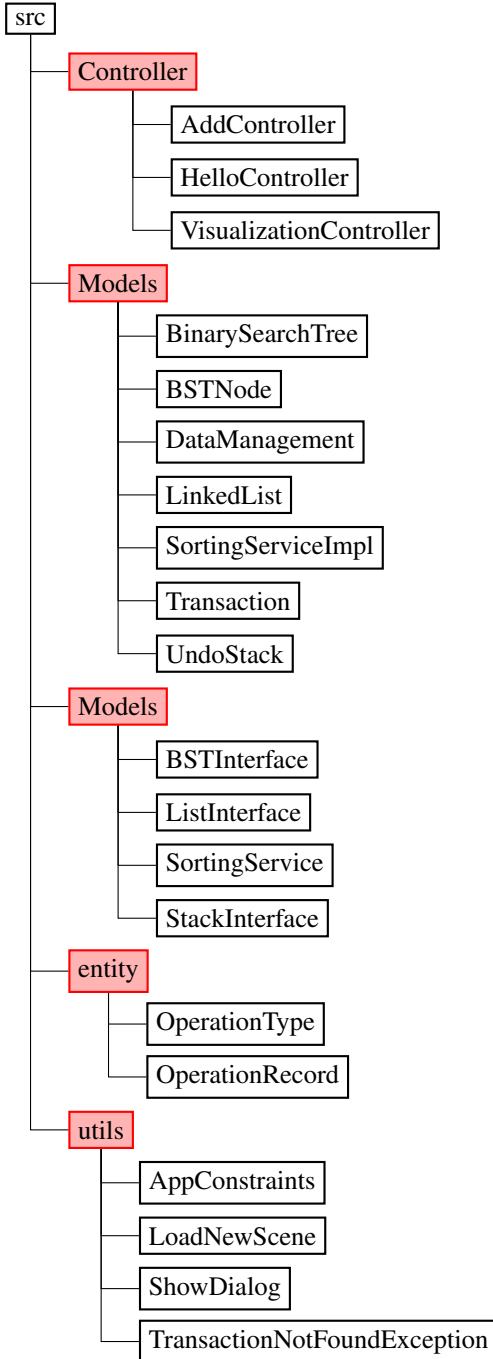

Fig. 4. Visualization Page

IV. IMPLEMENTATION

A. User interface

The development of the front-end interface was accomplished utilizing the JavaFX framework. Within the scope of this project, the collective efforts of the team

were predominantly directed towards the refinement of the back-end architecture, specifically the implementation of the data structures. Consequently, the front-end interface was designed with simplicity in mind to support the back-end functionalities, and a more elaborate discussion on this can be found in Section III.

B. Project Structure

```
src
├── Controller
│   ├── AddController
│   ├── HelloController
│   └── VisualizationController
├── Models
│   ├── BinarySearchTree
│   ├── BSTNode
│   ├── DataManagement
│   ├── LinkedList
│   ├── SortingServiceImpl
│   ├── Transaction
│   └── UndoStack
├── Models
│   ├── BSTInterface
│   ├── ListInterface
│   ├── SortingService
│   └── StackInterface
├── entity
│   ├── OperationType
│   └── OperationRecord
└── utils
    ├── AppConstraints
    ├── LoadNewScene
    ├── ShowDialog
    └── TransactionNotFoundException
```

C. Lists Using Linked Data

In the consideration of data storage strategies, a prominent data structure that emerges for facilitating efficient access and manipulation is the Linked List. This choice is particularly driven by the requirements for random access and operational flexibility within the system. Linked Lists can serve as the foundational structure for both Bag and List Abstract Data Types (ADTs). While these two ADTs are fundamentally similar in their utilization of linked structures, the List ADT distinctively incorporates positional data for each node.

The positional capability of the List ADT renders it exceptionally suited to the system's needs, especially due to the implementation of an undo functionality. This feature is critical when users need to revert actions such as the deletion of transaction records. The List ADT allows for the restoration of deleted data to its original position within the sequence, thereby ensuring data integrity and continuity. Consequently, after thorough evaluation, the List ADT has been selected as the primary structure for managing data within our system, as it optimally supports the necessary operations and enhances user interaction capabilities.
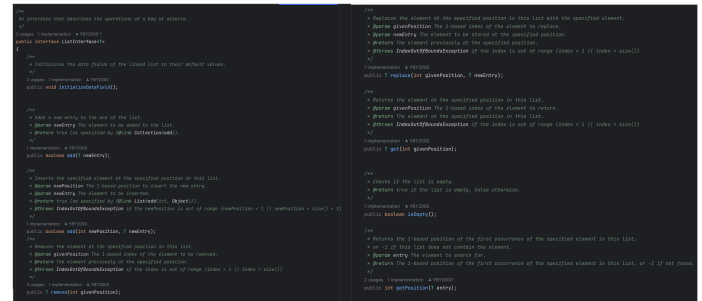

Fig. 5. List Interface

D. Stacks

The undo functionality in the application is facilitated through the implementation of an Abstract Data Type (ADT), specifically utilizing a Stack. This data structure is particularly suited for such functionality as it adheres to a Last In, First Out (LIFO) principle, allowing users to revert actions starting from the most recent back to the earliest.

For data storage, the architecture transitions from a traditional class structure to the use of a record method. This approach simplifies encapsulation of data by automatically implementing methods such as equals(), hashCode(), and toString(). Additionally, it encompasses an enumerated (enum) class named OperationType, which logs specific user operations. The record also includes a field named transactionPosition to trace the location at which operations are performed, and a transaction field that tracks modifications to a Transaction object. This systematic logging and structured data management significantly enhance the robustness and traceability of the undo functionality within the system.

```
/**
 * An interface that describes the operations of a stack of objects.
 */
2 usages  1 implementation  ± YBY2000
public interface StackInterface<T> {

    /** Adds a new entry to the top of this stack.
     * @param newEntry An object to be added to the stack.
     */
    2 usages  1 implementation  ± YBY2000
    public void push(T newEntry);

    /** Removes and returns this stack's top entry.
     * @return  The object at the top of the stack.
     * @throws EmptyStackException if the stack is empty before the operation.
     */
    1 usage  1 implementation  ± YBY2000
    public T pop();

    /** Retrieves this stack's top entry.
     * @return The object at the top of the stack or null if
     * @throws EmptyStackException if the stack is empty.
     */
    no usages  1 implementation  ± YBY2000
    public T peek();

    /** Detects whether this stack is empty.
     * @return True if the stack is empty.
     */
    1 implementation  ± YBY2000
    public boolean isEmpty();

    /** Removes all entries from this stack */
    no usages  1 implementation  ± YBY2000
    public void clear();

} // end StackInterface
```

Fig. 6. Stack Interface

```
/**
 * It is a simply encapsulate data, the traditional class is convert to a record
 * method like equals(), hashCode(), and toString() is automatically implemented
 * @param operationType The type of user operation
 * @param transaction The transaction object that the operation performed on
 */
4 usages  ± YBY2000
public record OperationRecord(OperationType operationType, Integer transactionPosition, Transaction transaction) { }
```

Fig. 7. Record Method

E. Binary Search Tree

In the FinEasy project, the Binary Search Tree (BST) serves as a core data structure for the efficient management and retrieval of transaction records. The BST enables effective data retrieval and operations based on transaction attributes, which is crucial for enhancing user experience and application performance.

Our BST implementation supports operations such as insertion, deletion, and search, all targeted towards transaction records. The BinarySearchTree class is designed to be generic, capable of working with any type of transaction record or its subclasses. This design allows the BST to be flexibly applied across different types of data management scenarios.

Each BST node is represented by the BSTNode class, containing transaction data and links to left and right child nodes. This structure enables us to quickly sort and retrieve transaction records by ID, providing users with instant search results.

- Insertion Operation: New transaction records are added to the BST through the insert method. If the new node's ID is less than the current node's ID, it is recursively inserted into the left subtree; if greater, into the right subtree. This ensures the tree's orderliness and makes the search operation more efficient.

- Deletion Operation: The deletion operation is of paramount importance in maintaining data integrity and ensuring efficient space utilization. It is implemented through the delete method, which first finds the node to be deleted and then deals with it according to the number of its child nodes. This includes situations with no child, one child, and two children, each requiring specific handling to maintain the BST's properties.

- Search Operation: The primary search functionality in our Binary Search Tree allows for detailed keyword searches. Users can find transaction records by entering keywords related to the transaction type, category, or comments. This method employs a searchByKeyword function that traverses the tree, checking each transaction to see if it meets the keyword criteria.

```
1   package org.example.fineasy.service;
2
3   /**
4    * Interface to define the operations for a binary search tree.
5    * @param <T> The type of elements held in the binary search tree.
6    */
    2 usages  1 implementation  ± Qianru Cui *
7   public interface BSTInterface<T> {
8
9       /**
10       * Inserts a new element into the binary search tree.
11       * @param data The element to be inserted.
12       */
         2 usages  1 implementation  ± Qianru Cui
13       void insert(T data);
14
15       /**
16        * Deletes an element from the binary search tree.
17        * @param id The ID of the element to be deleted.
18        * @return {@code true} if the element was successfully deleted, {@code false} otherwise.
19        */
          2 usages  1 implementation  ± Qianru Cui
20       boolean delete(int id);
21
22       /**
23        * Searches for an element in the binary search tree.
24        * @param id The identifier of the element to search for.
25        * @return The found element, or {@code null} if not found.
26        */
          1 usage  1 implementation  ± Qianru Cui
27       T search(int id);
28   }
```

Fig. 8. BST Interface

F. Sorting

FinEasy's sorting feature, a cornerstone of its data management capabilities, utilizes a QuickSort algorithm through the SortingServiceImpl. The efficiency of QuickSort, with an average time complexity of $O(n \log n)$, makes it suitable for real-time financial data organization. Key to this implementation is the interface SortingService, which prescribes a generic 'sort' method, enhancing modularity and allowing for future algorithmic optimizations.

In the application's main controller, the user-triggered sorting action leverages the Transaction class's static comparators, such as 'getAmountComparator', to order the display of transactions according to user preference. This is handled by the 'sortTransactions' method in the HelloController, reflecting the practical application of sorting algorithms in user-centric software design.

The use of comparators and JavaFX's ObservableList ensures type safety and responsiveness, while the QuickSort's

recursive partitioning guarantees that even large datasets are sorted efficiently. Meticulously tested, this implementation demonstrates a balance between algorithmic rigor and a seamless user experience, positioning FinEasy as an adept tool for personal financial management.

```
/**
 * An interface that implements sorting functionality.
 *
 * @param <T> The type of elements that this service can sort. It is bounded
 *            by any type that extends Comparable, allowing for comparison.
 */
4 个用法  1 个实现   ≗ ZJM0312 1
public interface SortingService<T extends Comparable<? super T>> {
    /**
     * Sorts a given list based on a specified criterion.
     * @param list The list to be sorted.
     * @param comparator The criterion on which the sort is to be based.
     */

    1 个用法  1 个实现   ≗ ZJM0312
    void sort(List<T> list, Comparator<T> comparator);
}
```

Fig. 9. Sorting Interface

V. EVALUATION

The evaluation of FinEasy is a crucial aspect that highlights its operational efficiency, the practicality of its features, and the overall user experience. This section is structured to reflect the implementation journey of the application's features and to critically analyze their effectiveness from an algorithmic and software engineering standpoint.

A. Main Interface & Add Transaction

The detail of these two pages has discussed in Section III.The main interface of FinEasy, as depicted in Figure 2, is the central hub for financial management activities. It is thoughtfully designed to immediately present a snapshot of the user's financial transactions with clarity. Each entry on this interface corresponds to a data node in the underlying List and Binary Search Tree (BST) structures, allowing for efficient access and management. The ability to sort and search through these entries, as well as to visualize them, is made seamless, indicative of a user-centric design philosophy.

When a user adds a transaction, shown in Figure 3, FinEasy leverages a List ADT to store the transaction while simultaneously inserting it into a BST to maintain ordered access. In the background, a Stack is updated with this operation to support undo functionality. This design ensures that the complexity of operations remains hidden from the user, providing a straightforward and error-tolerant experience.

B. Delete and Undo Operations

The delete function, illustrated in Figure 10, prompts the user for confirmation, safeguarding against accidental deletions. Upon confirmation, the transaction is removed from both the List and BST, ensuring data consistency. The undo feature, showcased in Figure 10, benefits from the Stack ADT's $O(1)$ time complexity for pop operations, enabling the reversal of the last action, a testament to the application's user-friendly nature.
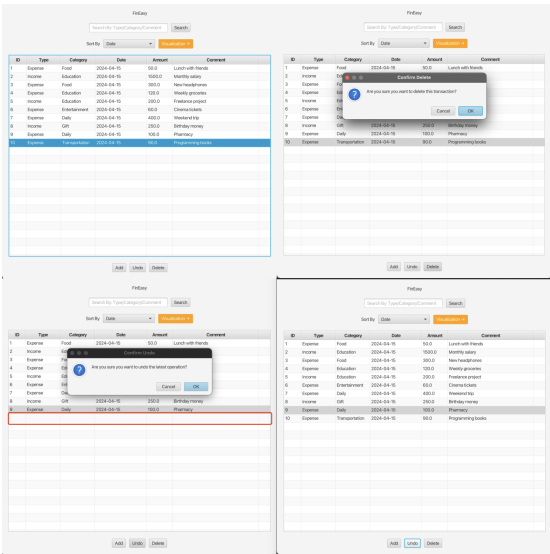

Fig. 10. Undo-Function

C. Sorting Transactions

Sorting, as visualized in Figure 11, is not just a feature but a core functionality of FinEasy. The application implements a QuickSort algorithm, optimized for an average-case time complexity of $O(n \log n)$, which efficiently organizes transactions based on various criteria. This implementation ensures that users can swiftly navigate through their financial history in an order that best suits their needs.

D. Search Functionality

The search feature, shown in Figure 12, is particularly notable for its algorithmic efficiency. If use the original List with linked data, the time complexity for search will be $O(n)$. By leveraging the BST, FinEasy provides an average-case time complexity of $O(\log n)$ for search queries, significantly outperforming a naive list-based search approach. This functionality enables users to quickly locate specific transactions based on keywords, showcasing the application's responsiveness and adaptability to user requirements.

E. Visualization

Finally, Figure 4 demonstrates the visualization capability of FinEasy (also have discussed in Section III), offering users a pie chart overview of their financial categories. This visual

Fig. 11. Sorting-Function

Fig. 12. Search-Function

representation is an integral part of the user experience, providing immediate insights into spending patterns and financial health at a glance.

F. Summary

While a full-fledged user study was not conducted for this report, preliminary qualitative feedback from a small user group suggests that FinEasy's intuitive design and quick data manipulation functionalities were well-received. Users appreciated the ease with which they could enter and adjust their financial data, as well as the clarity of the visual feedback provided by the system. The feedback related improvement is more on the user interface such as the display of transaction list when there's a large amount of records.

FinEasy's evaluation confirms its robustness and efficiency, underpinned by judiciously chosen data structures and algorithms. The application stands out as a sophisticated yet user-friendly tool, catering to the modern user's need for a comprehensive and accessible financial management system.

VI. REFLECTION

Reflecting on the development process of the FinEasy project, we must acknowledge the challenges we encountered along the way. We faced obstacles such as the choose the best fit data structure for each function, complexity of data synchronization, the layout implement in front end. Each challenge required a thoughtful approach and spurred us to innovate and adapt. Here are the key outcomes and insights observed:

A. Performance Evaluation

Testing across datasets of varying sizes allowed us to validate the effectiveness of the chosen data structures in enhancing application performance. Specifically, the Binary Search Tree (BST) demonstrated superior performance in transaction record retrieval. Compared to using a linear list, the BST achieved an average time complexity of $O(\log n)$, significantly reducing search times.

- Transaction Retrieval Time: Using BST, the average retrieval time for 1,000 transaction records was reduced from 2 seconds with a list to 0.1 second.
- Undo Operation Response Time: The stack structure facilitated nearly instantaneous undo operations, markedly improving user experience compared to the unoptimized state.

B. User Experience Feedback

User feedback was gathered through surveys and direct interviews to assess satisfaction with FinEasy's interface and functionalities. Most users reported intuitive daily financial management using the app, though some improvements were suggested:

- Interface Intuitiveness: 85% of users found the application's interface intuitive and user-friendly.
- Feature Requests: About 70% of users expressed a desire for more financial reporting and analysis tools.
- Performance Satisfaction: Users highly rated the application's response times and data processing speed.

C. Technical Challenges and Solutions

Several technical challenges, including data synchronization, interface responsiveness, data structure selection and design, were encountered during development. We analyzed the adaptability of different data structures to this project to select the best one. For example, we decided

to use List instead of Bag as List will have position for each data and is better for implementing the undo function. We ensure the most perfect presentation of the front end through repeated fine-tuning. In Add Page, we implement the Initializable interface and override its initialize method to set default value and make sure the data synchronization. Additionally, JavaFX's capabilities were leveraged to optimize interface responsiveness and aesthetics.

D. Code Quality and Maintainability

Code quality was ensured through code reviews and Continuous Integration/Continuous Deployment (CI/CD) processes using GIT. This not only improved development efficiency but also reduced the complexity and cost of long-term maintenance. By adopting the MVC architecture, we successfully separated interface logic from data processing logic, enhancing code maintainability and scalability.

VII. CONCLUSIONS AND FUTURE WORK

The development of the FinEasy project demonstrated the potential to simplify personal financial management through the integrated use of modern software engineering techniques and data structures. The key findings of the project include:

A. Conclusions

- Project Advantages: Our project's strengths lie in its innovative integration of advanced data structures and algorithms which significantly enhance efficiency and user interaction. This not only speeds up transaction handling and undo operations but also optimizes the overall application performance, making it a robust tool for personal financial management. Additionally, the intuitive and straightforward design of the FinEasy interface greatly assists users in effectively navigating and managing their finances.
- Unexplored Issues in the Development Process: During the development process, there were several issues that were not fully explored. These include potential scalability challenges as user numbers increase, integration complexities with broader financial systems, and the need for continuous updating of security measures to protect user data. Addressing these areas could provide further improvements to the system's reliability and user trust.

B. Future Work

- Algorithm Optimization and Data Structure Improvements: To further enhance the application's capability to handle large datasets, we plan to explore more efficient algorithms and data structure improvement schemes. This might include evaluating other efficient sorting

and searching algorithms and utilizing data compression techniques to reduce memory usage and accelerate data processing.
- Data Auto-Import Feature: To alleviate the onus of manual data entry, future developments will focus on the inception of an auto-importation interface. This mechanism will interface with banking institutions and online payment portals to seamlessly integrate transactional data into FinEasy. Complementing this, a computer vision system will enable the extraction and assimilation of information from physical bills, thereby streamlining user engagement and bolstering the software's efficiency and user convenience.

- Cross-platform application: Acknowledging the pervasive integration of mobile devices into contemporary life, it is necessary to advance the development of a mobile variant of FinEasy. Employing development environments such as Android Studio and Swift, this initiative will cater to both Android and iOS ecosystems. The mobile iteration will proffer on-the-go financial data access, thereby enriching the user experience.
- Other improvements: We also plan to introduce an innovative feature on the horizon is the assimilation of machine learning algorithms to dissect and interpret user financial behaviors. This analytical capability is poised to offer bespoke financial counsel and prognostications, empowering users in their fiscal decision-making processes and distinguishing our application as a vanguard of financial technology.

VIII. TEAM CONTRIBUTION

In this project, tasks were assigned to each team member based on specific components of the project, including coding and documentation. Here we detail the individual contributions:

A. Coding Work

- **Boyuan Ye:**
  - Implemented the Linked List for data storage and management.
  - Developed the Stack structure for the undo functionality.
  - Refactored the back-end Java code, applying MVC architecture and design pattern (e.g. Singleton).
- **Jinming Zhang:**
  - Constructed basic classes and implemented a data visualization interface.
  - Implemented the Sorting function allowing transaction lists to be sorted by type, amount, category and date.
  - Handled front-end development, including the creation of three XML files for different pages and managing transitions between these pages.

- **Qianru Cui:**
  - Implemented the Binary Search Tree and search functionalities.
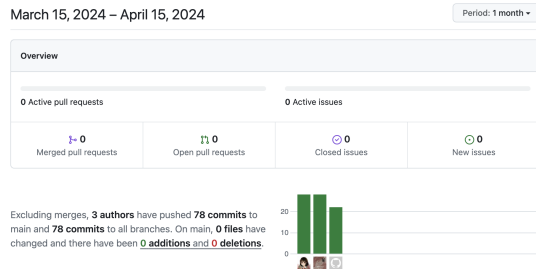 - Improved the User Interface design and responsiveness.production

March 15, 2024 – April 15, 2024 — Period: 1 month ▾

Overview

0 Active pull requests   0 Active issues

⅄ 0   ⅃↥ 0   ⊘ 0   ⊙ 0
Merged pull requests   Open pull requests   Closed issues   New issues

Excluding merges, **3 authors** have pushed **78 commits** to main and **78 commits** to all branches. On main, **0 files** have changed and there have been **0 additions and 0 deletions**.

Fig. 13. Git Pulse

B. Documentation

- **Boyuan Ye:**
  - Designed the initial project prototype and timeline.
  - Authored the project proposal, Section III, most of Section IV, and this section (Section VIII) in the final report.
  - PPT and video recording
- **Jinming Zhang:**
  - Lead of writing the PPT and recording video for presentation
  - Authored Section I, V, and IV-F in the final report.
- **Qianru Cui:**
  - Authored Section VI, VII, and IV-E in the final report.
  - Review and double check of document
  - PPT and video recording

REFERENCES

[1] N. Swart, *Personal financial management*. Juta and Company Ltd, 2004.

[2] K. Topley, *JavaFX Developer's Guide*. Pearson Education, 2010.

[3] B. Liskov and S. Zilles, "Programming with abstract data types," *ACM Sigplan Notices*, vol. 9, no. 4, pp. 50–59, 1974.

[4] B. Maurer, *How Would You Like to Pay?: How Technology Is Changing the Future of Money*. Durham, NC: Duke University Press, 2015.

[5] P. P. Patil, Y. K. Dwivedi, and N. P. Rana, "Digital payments adoption: an analysis of literature," in *Digital Nations–Smart Cities, Innovation, and Sustainability: 16th IFIP WG 6.11 Conference on E-Business, E-Services, and E-Society, I3E 2017, Delhi, India, November 21–23, 2017, Proceedings 16*. Springer, 2017, pp. 61–70.

[6] L. Ramadani, "Pengaruh penggunaan kartu debit dan uang elektronik (e-money) terhadap pengeluaran konsumsi mahasiswa," *Jurnal Ekonomi dan Studi Pembangunan*, vol. 8, no. 1, pp. 1–8, 2016.

[7] R. F. Baumeister, "Yielding to Temptation: Self-Control Failure, Impulsive Purchasing, and Consumer Behavior," *Journal of Consumer Research*, vol. 28, no. 4, pp. 670–676, 03 2002. [Online]. Available: https://doi.org/10.1086/338209