



KENNESAW STATE UNIVERSITY

AI7993 - AI Capston
Section W01
Spring Semester
4/28/2025

[AC-1] Bill Splitter Mobile App – Share the tab

Professor

Dr. Arthur Choi

Individual Assignment

Name

Mira Saad

[Website Link](#)

[Github Link](#)

Contents

1. Introduction	3
2. Requirements.....	3
2.1 Functional Requirements.....	3
2.2 Non-Functional Requirements.....	3
3. Analysis	4
4. Design	5
5. Development	7
5.1 Initial Prototype Using React Native and Expo Go.....	7
5.2 Exploring an AI-Powered Receipt Reader	7
5.3 Transition to Flutter with Native OCR Integration.....	8
5.4 Development Highlights and Architecture.....	8
5.5 Final Outcome	8
6. Challenges.....	9
7. Version Control	10
8. Summary	10
9. Appendix	12
9.1 Project Plan Summary.....	12
9.2 Screen Mockups & Information Flow	13
9.2.1 User Flow Overview	13
9.2.2 Screen Mockups	13
9.3 Technologies & Libraries.....	16
9.4 Data Flow Diagram	16

1. Introduction

In today's fast-paced social environments, dining out with friends is common, but splitting the bill fairly and efficiently remains a consistent source of frustration. This project introduces a mobile application designed to streamline and simplify the bill-splitting process by using Optical Character Recognition (OCR) technology. By allowing users to snap a picture of a restaurant receipt, the app automatically extracts itemized data, assigns items to individuals, and calculates how much each person owes. The goal is to eliminate the hassle of manual calculations, reduce errors, and provide a smooth, user-friendly experience that works across both iOS and Android platforms. Key objectives include accurate text recognition, seamless UI design, cross-platform compatibility, and secure data handling. Ultimately, this app aims to redefine how groups share expenses when dining out, making it faster, fairer, and far more convenient.

2. Requirements

2.1 Functional Requirements

The mobile application will offer users the ability to capture a photo of a physical restaurant receipt using their smartphone camera or upload an image from their gallery. Upon capturing the image, the app will utilize Optical Character Recognition (OCR) technology to extract and parse text from the receipt. This process will identify itemized entries, prices, subtotals, taxes, and totals. The extracted data will then be structured so that users can assign specific items to individuals in the group. Each item can be linked to one or more users, and the app will handle shared items accordingly.

Once the assignment is complete, the application will calculate the amount owed by each person, factoring in taxes and tips. It will present this calculation in a clear, summarized format for review and confirmation. To address potential OCR inaccuracies, the app will provide manual override options that allow users to edit item names, prices, and other details directly within the interface. All past transactions, including receipt data and user contributions, will be stored locally and automatically synced to the cloud when internet connectivity is available. Additionally, the app will provide a history log of past receipts, allowing users to revisit and manage their prior bills. The interface will be designed to support intuitive navigation, smooth assignment workflows, and responsive user interactions across both Android and iOS platforms.

2.2 Non-Functional Requirements

The app will be built using the Flutter framework to ensure cross-platform compatibility and a consistent user experience across iOS and Android devices. Performance is a critical consideration; the app will be optimized to process images and perform calculations within a few seconds, even on lower-end smartphones. Efficient image compression and memory management strategies will be used to support smooth performance. The app must also remain responsive and

fully functional offline, with the ability to sync data to a cloud backend like Firebase once a network connection is re-established.

The user interface will follow Material Design principles, providing an aesthetically pleasing, easy-to-navigate layout that accommodates users with varying levels of technical proficiency. Accessibility features such as adjustable font sizes and screen reader compatibility will be included to support users with disabilities. The system will be designed with scalability in mind, allowing for the addition of future features like mobile payments, multi-currency support, or recurring bill management.

Security and privacy are essential to the app's design. All user data, including receipt content and calculated bill shares, will be encrypted both in transit and at rest. The app will require minimal permissions (camera, storage, internet) and will request them only when necessary to ensure trust and user comfort. Finally, modular and maintainable system architecture will be used to support easy debugging, testing, and future upgrades.

3. Analysis

The development of this mobile application addresses a common social inconvenience: the complexity of splitting a restaurant bill among friends or groups. In many dining situations, manually calculating who owns what can lead to confusion, errors, or even discomfort within the group. Traditional solutions, such as mental math, manual note-taking, or generic calculators, lack the accuracy, efficiency, and fairness that users expect in a digital age. This project seeks to resolve these issues through the integration of OCR technology with a clean, intuitive user interface developed in Flutter.

At the core of the solution is the use of Optical Character Recognition to automate data entry by extracting structured data from unstructured receipt images. This eliminates the need for users to manually transcribe item names and prices, significantly reducing time and potential for error. However, OCR implementation introduces technical challenges such as handling variations in receipt formats, distorted or low-quality images, and different font styles or handwriting. Thus, the accuracy of the receipt parsing process will heavily depend on the robustness of the OCR engine and the app's ability to handle edge cases like missing data or incorrectly recognized characters.

The target users of the app are groups of people dining together, often in social settings, who value speed, simplicity, and fairness in splitting expenses. Therefore, the app must offer a user experience that is fast, easy to understand, and non-intrusive. The interface should cater to both tech-savvy users and those less familiar with mobile apps. This implies the need for accessible design, including clear instructions, flexible input options, and supportive features like text-to-speech or large-text modes.

From a technical perspective, the use of Flutter provides a powerful foundation for rapid development and deployment across platforms, ensuring visual and functional consistency between Android and iOS versions. Flutter's widget-based structure also supports a modular

architecture that aligns well with the app's needs for separation of concerns, particularly between OCR processing, business logic, and the user interface.

Key risks identified during analysis include dependency on third-party OCR services, which may limit customization or result in service interruptions. Furthermore, device limitations, such as low-resolution cameras, poor lighting, or outdated operating systems, can negatively impact OCR performance. To mitigate these risks, the app must include fallback mechanisms such as manual editing of receipt data and user prompts for retaking unclear images.

In conclusion, the analysis confirms both the demand for and feasibility of the proposed application. It highlights the technical opportunities provided by Flutter and OCR, while also recognizing potential usability, accuracy, and privacy challenges. These findings inform the subsequent design and development phases, ensuring that the app is built with a clear understanding of user needs, system capabilities, and environmental constraints.

4. Design

The design of the mobile application is centered on creating a modular, scalable, and user-friendly system that efficiently utilizes OCR technology to automate the process of splitting restaurant bills. The application architecture follows a layered and modular design pattern, dividing core functionality into distinct components: the OCR Processing Module, Bill Calculation and Share Assignment Module, User Interface Module, and Data Storage & Synchronization Module. Each module is developed to operate independently yet cohesively, ensuring maintainability, testability, and future extensibility.

At the foundation, the app is developed using Flutter, a cross-platform framework that enables consistent and responsive UI on both Android and iOS devices. Flutter's widget-based architecture is particularly suited to the modular design approach, allowing UI components to be built, reused, and maintained efficiently. The UI follows Material Design principles, prioritizing clarity, accessibility, and minimalism. Users are guided through a linear workflow: capturing or selecting a receipt image, reviewing and assigning extracted items, and reviewing the calculated individual shares. The interface includes clear call-to-action buttons, visual feedback on assignments, and support for editing extracted data manually if needed.

The OCR module is responsible for analyzing receipt images and converting printed text into machine-readable data. It employs preprocessing techniques such as image enhancement, rotation correction, and cropping to improve OCR accuracy in varying conditions. Once the receipt is parsed, the data is structured into a JSON-like format containing item names, prices, tax, and total. This structured data is then sent to the Bill Calculation and Share Assignment module.

The Bill Calculation module handles all logic related to distributing costs among users. It supports assigning items to individuals, splitting shared items, and automatically calculating taxes and tips proportionally. The module is designed to be flexible and responsive, handling changes

in assignments and recalculating values in real-time. Edge cases, such as unassigned items or duplicate assignments, are addressed through built-in validation rules and visual alerts in the UI.

In terms of scalability and maintainability, the app follows separation of concerns, with UI, data, and logic components kept distinct. This allows for easy updates and future enhancements, such as the addition of a payment integration system, support for handwritten receipts, or AI-driven error correction. Each module communicates through well-defined interfaces, allowing developers to replace or extend components with minimal disruption to the overall system.

In summary, the app's design combines technical robustness with a seamless user experience. It leverages Flutter's cross-platform capabilities, integrates powerful OCR tools, and maintains a clean modular structure that ensures adaptability and growth. The design philosophy emphasizes automation, accuracy, and simplicity, creating a reliable tool for simplifying group dining experiences.

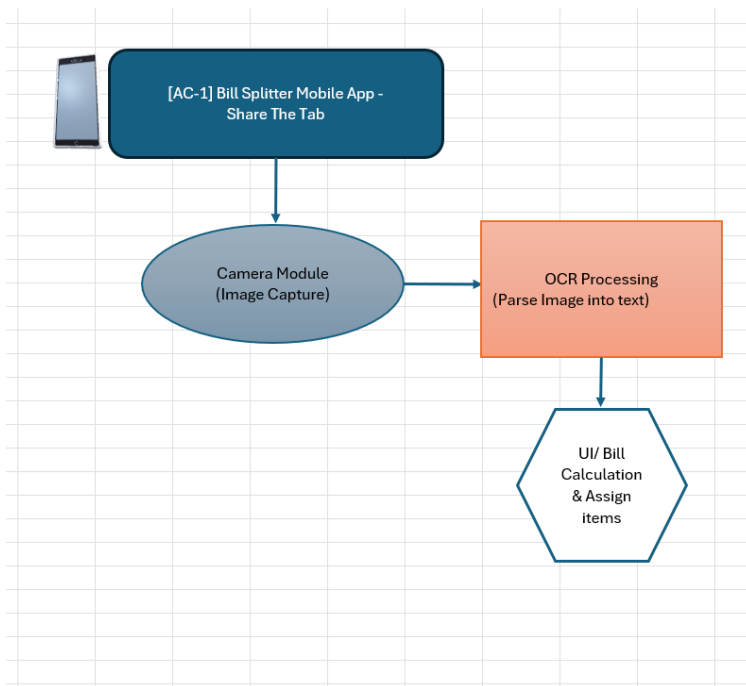


Figure (1): Software Architecture for Receipt-Based Bill Splitting Mobile App.

This figure represents the modular architecture of the mobile application designed to automate receipt scanning and bill splitting among friends. The architecture follows a layered and component-based design, starting with the camera module for capturing receipt images, followed by OCR processing and parsing to extract structured data. It includes a robust error-handling system, a core bill calculation engine for assigning costs and

tips. The UI layer provides an intuitive, responsive interface using Material Design principles, ensuring accessibility across iOS and Android platforms. The architecture supports scalability, offline-first usage, and future extensibility such as payment integration.

5. Development

The development process for the bill-splitting app was iterative and experimental, involving several major shifts in tools and technology before arriving at the final solution. I aimed to build a cross-platform mobile application that could use OCR to extract text from restaurant receipts and help users split the bill fairly and quickly. Along the way, I explored different frameworks and technologies, each contributing valuable insights and lessons.

5.1 Initial Prototype Using React Native and Expo Go

- Framework chosen: React Native
- Goal: Build a cross-platform application quickly and efficiently.
- Tool used: Expo Go for fast prototyping.

I started the project using React Native due to its wide adoption and support for cross-platform development. I used Expo Go in the early stages to simplify the development and testing process. However, I quickly encountered a significant limitation, Expo Go does not support native modules, which are essential for integrating OCR functionality. This blocked the ability to implement real-time text recognition, a core feature of the app.

5.2 Exploring an AI-Powered Receipt Reader

- Solution attempted: AI API to generate structured tables from receipt images.
- Implementation: Built a React Native front-end and integrated with an external AI service.
- Outcome: Abandoned due to reliability and security concerns.

In response to the Expo limitations, I explored an alternative method using a third-party AI-powered API to process images and return an itemized list of purchases. While this approach offered potential in theory, it struggled in practice. The AI model often returned inaccurate or incomplete results, particularly with unclear or cluttered receipts. Additionally, sending sensitive receipt data to external servers raised privacy concerns, and the delay in receiving results from the API affected the user experience. As a result, this method was ultimately deemed unsuitable for our needs.

5.3 Transition to Flutter with Native OCR Integration

- Framework switched to: Flutter
- OCR tool used: Google ML Kit
- Reason for change: Better native support, performance, and full access to device features.

To overcome the technical barriers, I made a strategic decision to switch to Flutter, a UI toolkit developed by Google. Flutter provided seamless access to native device features like the camera, a powerful on-device OCR engine. This gave us the flexibility and control I needed to build a robust, fast, and privacy-conscious OCR pipeline.

I designed a system where the user can take a photo of a receipt, and the OCR module extracts the text, which is then parsed into items and prices. From there, users can assign items to individuals, and the app calculates each person's total based on their selections, tax, and tips.

5.4 Development Highlights and Architecture

- Modular architecture for scalability and maintainability.
- Test-Driven Development (TDD) used to ensure quality and reliability.
- Material Design UI for a clean and consistent user experience across platforms.

The app was developed using a modular approach, with separate components for OCR, bill calculation, user interface, and cloud syncing. This made it easy to test, maintain, and expand features over time. I followed an Agile methodology, incorporating TDD for key features to reduce bugs and maintain high-quality code.

5.5 Final Outcome

After several iterations and careful consideration of various technologies, the final outcome was a fully functional, cross-platform mobile application developed using Flutter. The app integrates on-device OCR, allowing users to capture receipt images and extract itemized text efficiently and securely without relying on external servers. The user interface, designed with Flutter's Material Design system, provides an intuitive and visually consistent experience across both iOS and Android platforms. Users can assign items to individuals, and the app automatically calculates each person's share, including taxes and tips. The architecture supports offline functionality with local storage, while Firebase enables real-time cloud synchronization for multi-device access. The final solution delivers speed, accuracy, and ease of use, meeting the project's original goal of simplifying the bill-splitting process in group dining scenarios.

6. Challenges

Throughout the development of the bill-splitting application, I encountered a variety of technical and conceptual challenges that significantly influenced the direction and design of the final product. These challenges ranged from early technology limitations to the complexities of real-world OCR implementation and cross-platform compatibility.

One of the first major challenges arose during the initial prototyping phase with React Native. While React Native was initially appealing due to its widespread use and rapid development potential, it became clear that Expo Go, the tool being used to test and deploy the app, did not support the native modules required for integrating Optical Character Recognition (OCR) libraries. This limitation effectively blocked progress on one of the app's core features and forced to reconsider its development stack. Attempts to eject from Expo and use native modules directly were time-consuming and introduced new maintenance complexities.

Another significant challenge was encountered when exploring an AI-based solution to parse receipts. I tested an approach that involved uploading receipt images to an external AI API that would generate a structured table of items and prices. While promising in theory, the solution faced several practical issues. The accuracy of AI-generated results was inconsistent, particularly with low-quality or complex receipts. Additionally, this approach introduced latency issues and privacy concerns, as sensitive financial data had to be transmitted to third-party servers. These factors made it unsuitable for a seamless and secure user experience.

After transitioning to Flutter, I had to overcome a new set of obstacles, particularly around OCR integration and performance. While Flutter provided much better support for native functionality and access to the camera, working with OCR engines required fine-tuning to handle real-world scenarios such as blurry images, poor lighting, skewed text, and varied receipt layouts. Ensuring accurate data extraction across different types of receipts was a recurring challenge and required extensive testing and validation.

Furthermore, designing a user interface that was both powerful and intuitive proved to be a balancing act. The app needed to support advanced functions like item assignment and manual correction of OCR errors, while remaining accessible to users with varying levels of technical experience. Achieving this required multiple UI iterations and usability testing.

Cross-platform development also introduced compatibility concerns. Ensuring consistent performance, layout responsiveness, and device-specific behavior across both iOS and Android, especially on lower-end devices with limited camera quality or processing power, required careful optimization and testing.

Despite these obstacles, each challenge contributed to the refinement of the final product. The decisions made in response, such as switching frameworks, integrating robust OCR tools, and optimizing the UI, were critical to developing a high-quality, user-centered application. These challenges ultimately guided the project toward more practical, secure, and scalable solutions.

7. Version Control

To ensure efficient collaboration, maintain code integrity, and support the iterative development process, version control was managed using Git in conjunction with GitHub as the remote repository hosting service. Git provided a reliable and flexible way to track changes across the entire codebase, while GitHub facilitated distributed work, issue tracking, and continuous integration support.

The project followed a feature-branch workflow, where the main branch represented the stable, production-ready version of the application, and individual features or fixes were developed in separate branches. Each feature branch was named descriptively to clearly reflect its purpose. Developers created pull requests (PRs) when merging into the main branch, allowing for peer reviews and ensuring code quality through feedback before integration.

During the early stages of the project, multiple experiments were conducted using different branches to test the feasibility of various tools and technologies. For instance, branches were created to prototype React Native with OCR, and another to implement the AI-powered table generation using an external API. These experimental branches were later archived when transitioned to Flutter and finalized the OCR-based implementation.

To maintain code quality, pre-commit hooks and linting tools were implemented, especially for Flutter development using tools like flutter Lints and dart analyze. These ensured consistent formatting and reduced the number of syntax and logic errors in the codebase. GitHub Actions were configured to automate basic build checks and test runs on each pull request to detect issues early.

Commits followed a standardized convention to keep the history readable and structured. This practice helped during debugging, code audits, and milestone tagging. Version tags were periodically added to the repository to mark significant development stages such as the initial React Native prototype, the AI table generation trial, and the eventual stable Flutter release.

Backups of the repository were ensured by keeping it synchronized with GitHub, and branches were regularly merged, cleaned up, and documented to prevent clutter and confusion. Collaboration was further supported through GitHub Projects and Issues, which were used to organize tasks, prioritize bugs, and track progress across development sprints.

In summary, Git and GitHub provided a robust foundation for managing the evolving codebase, supporting collaboration, experimentation, and ultimately ensuring the delivery of a clean, maintainable, and well-documented Flutter-based application.

8. Summary

This project is set out to develop a mobile application that streamlines the process of splitting restaurant bills using Optical Character Recognition (OCR) technology. Through a user-friendly interface, the app allows users to capture images of receipts, automatically extract and organize itemized data, assign items to individuals, and calculate fair payment shares. The core objective

was to reduce the hassle and inaccuracies often associated with manual bill splitting, providing a faster, smarter, and more intuitive solution for group dining scenarios.

The development journey involved several stages of exploration and adaptation. Initially, React Native was chosen for its cross-platform capabilities; however, technical limitations, particularly the lack of OCR support in Expo Go, hindered its feasibility. This led to the exploration of an AI-powered approach using external APIs to generate item tables from receipt images. While innovative, the AI solution proved inconsistent in real-world scenarios and raised privacy and latency concerns. Ultimately, the project transitioned to Flutter, which offered superior native support, better performance, and easier integration with on-device OCR engines. This shift proved critical to achieving both functionality and stability.

The final application was built using modular architecture with well-separated components for OCR processing, bill calculation, user interface, and data storage. Flutter's flexibility enabled a consistent experience across both Android and iOS platforms. Agile methodologies and test-driven development (TDD) ensured continuous progress, adaptability, and high-quality code. Version control with Git and GitHub supported collaboration, experimentation, and structured project management throughout the development lifecycle.

In conclusion, the project successfully delivered a scalable and accessible mobile app that meets the need for accurate, efficient, and user-friendly bill splitting. The integration of OCR and thoughtful design choices ensures the app remains adaptable for future enhancements, including payment integration, better error handling, and AI-powered features. This outcome reflects a thoughtful balance between innovation, practicality, and user-centered design.

9. Appendix

9.1 Project Plan Summary

Phase	Duration	Key Activities	Deliverables
Requirements Gathering	Week 1	Define functional & non-functional requirements	Requirements Document
Prototyping	Weeks 2–3	React Native prototype; AI receipt parser tests	Prototype builds, Evaluation Report
Framework Transition	Week 4	Migration to Flutter, integration of Google ML Kit OCR	Flutter scaffold with OCR functionality
Core Development	Weeks 5–8	OCR parsing, item assignment, bill calculation modules	App Build
UI/UX Design	Weeks 7–8	Build intuitive UI using Material Design principles	Interactive UI components
Testing & Debugging	Weeks 9–10	Functional/unit testing, TDD, device compatibility	Stable Beta Version
Finalization & Deployment	Week 11	Cloud sync integration (Firebase), offline support	Production Build
Documentation	Week 12	Code documentation, user guide, final report	Complete Documentation Set

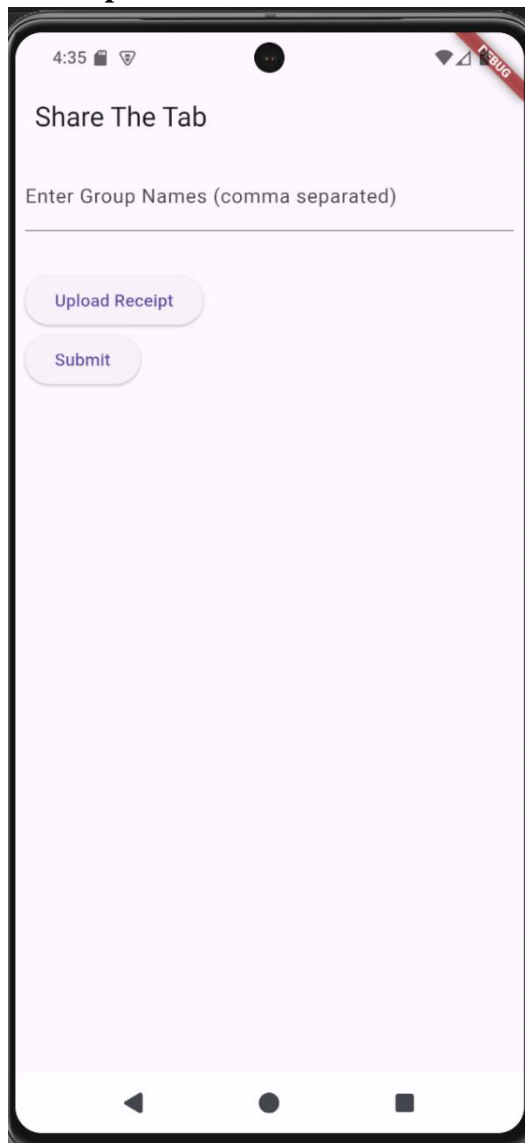
9.2 Screen Mockups & Information Flow

9.2.1 User Flow Overview

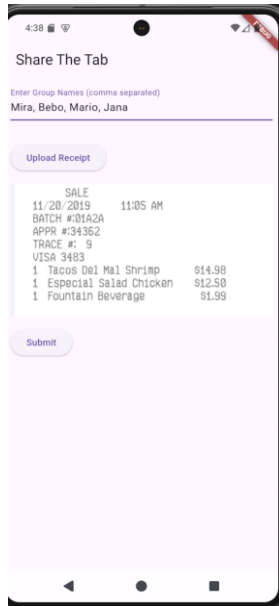
[Start] → [Capture or Upload Receipt] → [OCR Extracts Items] → [Assign Items to People] → [Review Totals] → [Save or Share Bill]

9.2.2 Screen Mockups

Mockup 0: Home Screen



Mockup 1: Receipt Capture Screen



4:38

Share The Tab

Enter Group Names (comma separated)

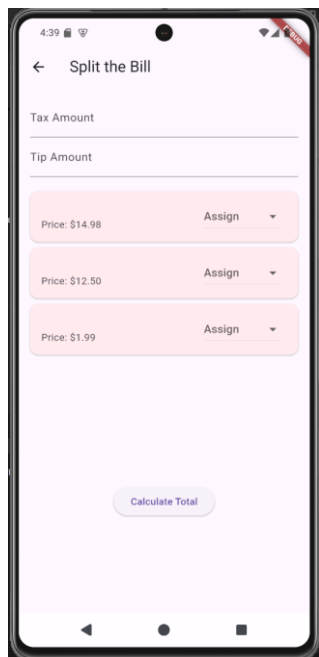
Mira, Bebo, Mario, Jana

Upload Receipt

SALE
11/20/2019 11:35 AM
BATCH #30142A
APPR #34362
TRACE #: 9
VISA 3483
1 Tacos Del Mar Shrimp \$14.98
1 Especial Salad Chicken \$12.50
1 Fountain Beverage \$1.99

Submit

Mockup 2 : Parsed Item View



4:39

← Split the Bill

Tax Amount

Tip Amount

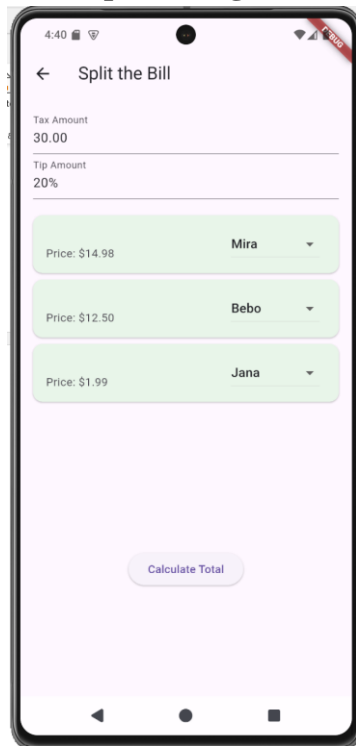
Price: \$14.98 Assign

Price: \$12.50 Assign

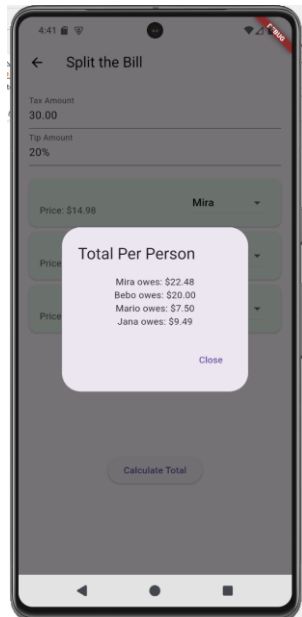
Price: \$1.99 Assign

Calculate Total

Mockup 3 : Assignment Screen



Mockup 4: Final Summary



9.3 Technologies & Libraries

- **Framework:** Flutter (cross-platform)
- **OCR Engine**
- **Version Control:** Git + GitHub
- **UI Design:** Flutter Material Design Widgets
- **Testing:** Flutter Test, Manual QA on devices

9.4 Data Flow Diagram

