CHAPTER V
CONCLUSION

5.1 Conclusions

Based on the research findings and comprehensive discussions regarding the development of the OCR-based financial management web application, the following conclusions are drawn:

1. **Implementation of Optical Character Recognition (OCR):** The implementation of Optical Character Recognition (OCR) technology utilizing the Python-based Tesseract engine successfully detects and extracts text components from shopping receipt images. However, this extraction performance is strictly bound to the characteristics of the tested primary dataset, consisting of 15 PNG receipt images from Superindo retail stores. The system exhibits a structural limitation where OCR performance is optimized exclusively for that specific brand's layout, producing blank outputs when processing receipts from other merchants. Furthermore, extraction success relies heavily on standardizing input image quality, requiring sufficient lighting, an orthogonal (non-tilted) orientation, a clean background, and precise cropping around the transaction area.

2. **Impact of Image Preprocessing:** Implementing various combinations of image preprocessing techniques using OpenCV and NumPy significantly improves image quality prior to OCR execution. Through proper preprocessing engineering, the baseline accuracy limitations of the Tesseract module were optimized to reach an extraction accuracy range of 82%–85%. Although initial experiments with PaddleOCR yielded a higher accuracy rate of 94%, the final architectural decision favored Tesseract-OCR. This choice represents a deliberate technical trade-off, as PaddleOCR requires substantial computational resources (resource-heavy) that triggered system crashes when deployed on resource-constrained server environments.

3. **Web-Based Financial Management Application Development:** The web-based financial management application was successfully developed through the integration of a multi-language programming architecture. The Python backend executing the Tesseract OCR module was connected to the core financial management logic built on PHP (Laravel framework). An interactive user interface was engineered using JavaScript, enabling automated transaction data extraction from Superindo receipts alongside visual financial reporting via pie charts and column charts. The full web application operates stably and responsively when deployed on an entry-level DigitalOcean VPS Droplet (1 vCPU Core, 1 GB RAM, 25 GB NVMe SSD, and 1 TB Bandwidth), having been developed locally on a hardware setup featuring an AMD Ryzen 3 5300U processor, 8 GB RAM, and a 512 GB SSD.

5.2 Recommendations for Future Work

For further research and system expansion, the following directions are recommended:

1. **Infrastructure Scalability and PaddleOCR Implementation:** Future research should upgrade the deployment server infrastructure by expanding the DigitalOcean VPS Droplet capacity (e.g., increasing memory to a minimum of 4 GB RAM or utilizing Dedicated CPU instances). Supported by higher computational resources and hardware budget allocation, future iterations can implement deep learning-based OCR engines such as PaddleOCR or CRAFT to achieve higher data extraction accuracy (exceeding 94%) without encountering system crashes.

2. **Dataset Expansion and Layout Generalization:** Subsequent studies should broaden both the dataset volume and variety beyond the single-merchant focus on Superindo receipts. Dataset expansion should encompass receipts from diverse modern retail chains, traditional stores, and handwritten notes. This improvement should be paired with adaptive text detection algorithms so that the system handles randomized or non-standard transaction layouts effectively.

3. **Mobile Platform Portability and Asynchronous Processing:** To enhance daily user experience, this web application should be extended to mobile

platforms (Android/iOS) or redesigned as a Progressive Web App (PWA), allowing users to scan receipts directly via smartphone cameras. Additionally, to handle batch image processing without blocking the main application thread, future backend architectures should incorporate asynchronous processing or message queues (such as Celery or Redis).