PaperFlow: Split Long Image for Printing

Course: CM3050 Mobile Development

Author: Yue Wu

Date: March 10, 2025

1. Concept Development

1.1 Background

Long-form text-based images have emerged as a prevalent medium on the internet, especially within the Chinese-speaking online community, driven by three key factors: 1) the widespread support for scrolling screenshots among smartphones in China; 2) the popularity of image stitching applications and article-to-image tools; and 3) the established practice of sharing long pictures instead of links/texts.

While this trend reflects user preferences and behaviours, it presents a significant challenge, particularly in the context of printing. For example, legal professionals often require hard copies of long-image evidence, such as unrolled WeChat chat logs, for court submissions.

A thorough search of available applications reveals a market gap. Current lightweight image-splitting tools only let users divide images into N equal parts, risking text truncation and forcing them to calculate the correct number of segments for proper printing aspect ratios. Alternatively, users must manually create multiple copies of the original image and crop each page individually, which is time-consuming and also makes it difficult to maintain uniform proportions.

This project proposes a targeted solution to effectively address the long image printing challenge.

1.2 Project Overview

Discover **PaperFlow**, the handy app that simplifies preparing long images for print. Perfect for lawyers and anyone dealing with scrolling screenshots, PaperFlow allows you to easily import photos and automatically split them into A4, Letter, or Legal-sized pages. Fine-tune split lines manually and export your documents in print-ready format with just a few clicks. Experience hassle-free printing like never before!

1.3 Core Features

- **Image Import**

Users can select an image from their local photo library and set splitting options, including page size and auto-split.

- **Automatic Splitting**

Automatically generates split lines based on user-selected proportions (A4, Letter, Legal).

- **Manual Adjustment**

Displays split lines on the original image, enabling users to manually add, delete, or adjust them.

- **Preview Interface**

Shows cropped images in a carousel for users to preview the final result.

- **Export Functionality**

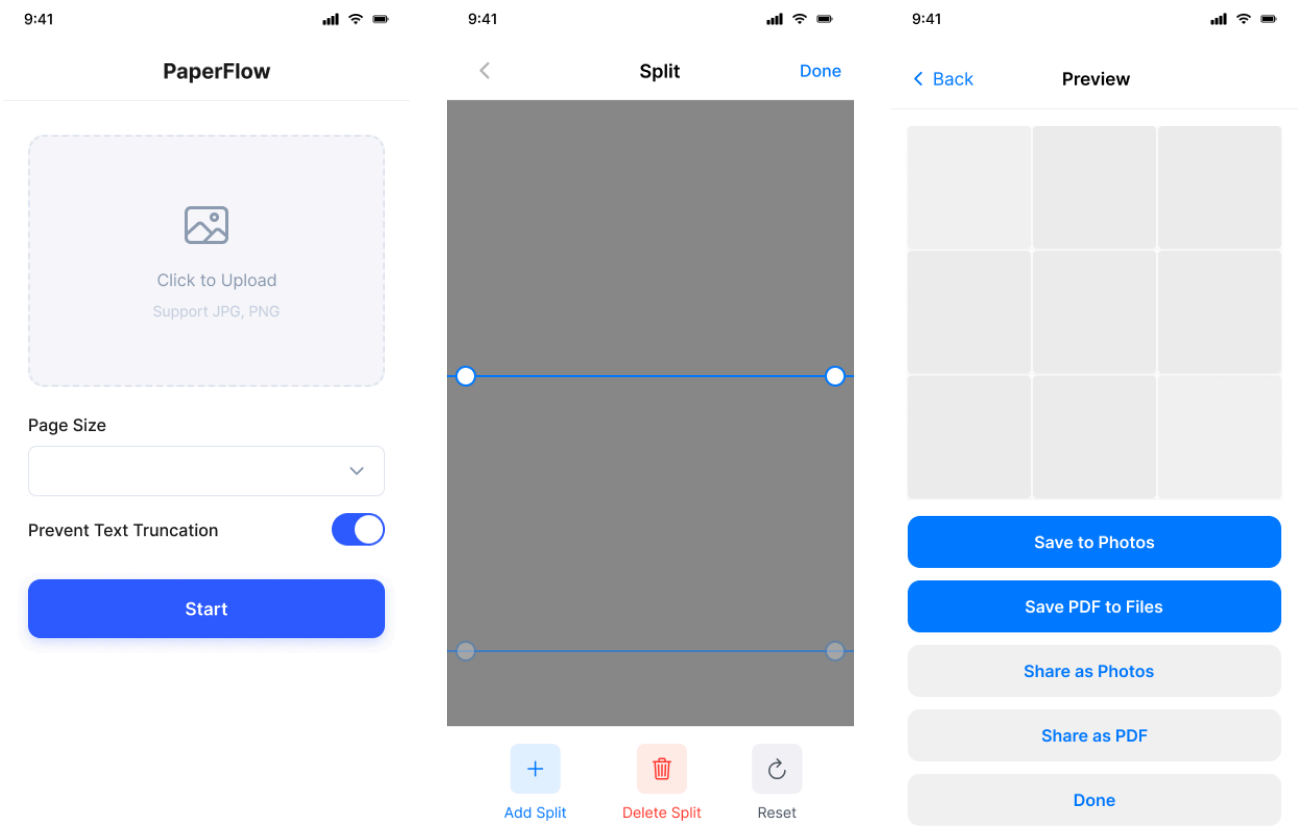
Lets users save the results as images or share them as a single PDF file.

2. Wireframing

For this project, I completed three rounds of wireframing. Each round was user-tested and refined based on user feedback.

2.1 Wireframe Version 1

In the initial wireframing round, I designed the app to have three main pages: 1) an image import page, 2) a split workshop page, and 3) a preview and export page. Users can import an image and set splitting options on the Home page, adjust split lines on the Split page, and preview and export the final result on the Preview page. This outlines the first version of the low-fidelity wireframe.



Wireframe Version 1

2.2 Wireframe Version 2

Based on the user feedback on Version 1, the following changes were made:

1. **Home Screen**

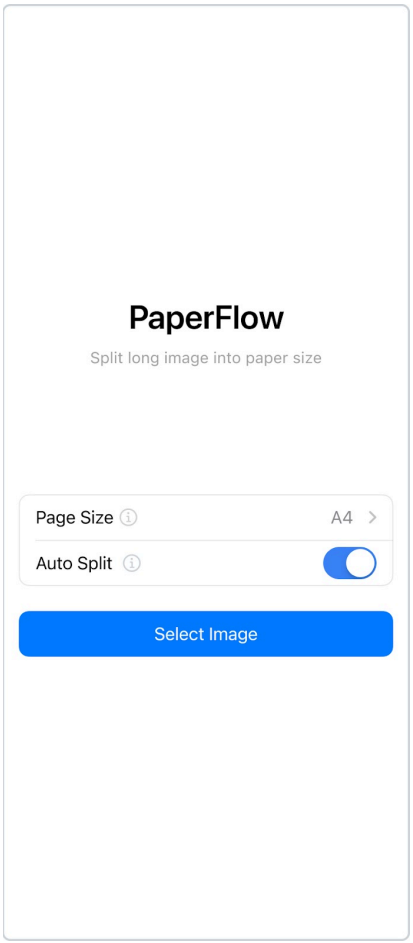
- The **image upload component** has been removed, and its functionality is now integrated with the start button. Once an image is selected, the app will automatically transition to the next screen.
- Due to privacy concerns regarding OCR text detection, the **'Prevent Text Truncation' feature** has been removed and replaced with an 'Auto Split' option.
- A small **question mark icon** has been added next to the options to provide users with interactive help. Clicking this icon opens an explanatory popup.
- The **app name** has been enlarged and repositioned for better visual alignment. A one-line description has been added beneath the app name to clarify its primary functionality.

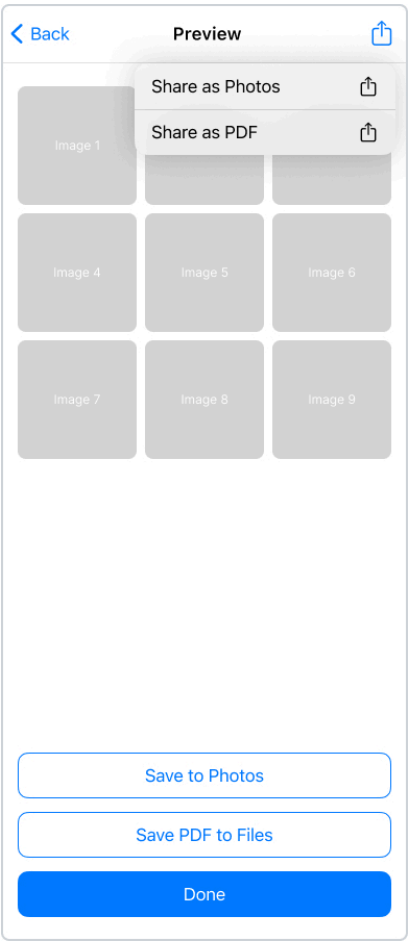
2. Split Screen

- A **loading indicator** has been added to improve the user experience during image processing.
- **Text indicators** have been added to explain the gesture-based functionality more clearly.
- A **page number indicator** has been implemented so users can track which page they are on.
- The **drag and delete buttons** have been minimised and placed next to each line for easier access.
- The number of **bottom buttons** has been reduced from three to two.
- **UI text** has been updated for greater clarity.

3. Preview Screen

- The **share functionality** has been moved to the top-right corner to align with iOS design conventions.
- The number of **bottom buttons** has been reduced to eliminate visual clutter.





2.3 Wireframe Version 3

After more user tests, more updates have been made:

1. **General Changes**

- The **colour palette** and **font pairing** have been selected to ensure consistency across the app.

2. **Home Screen**

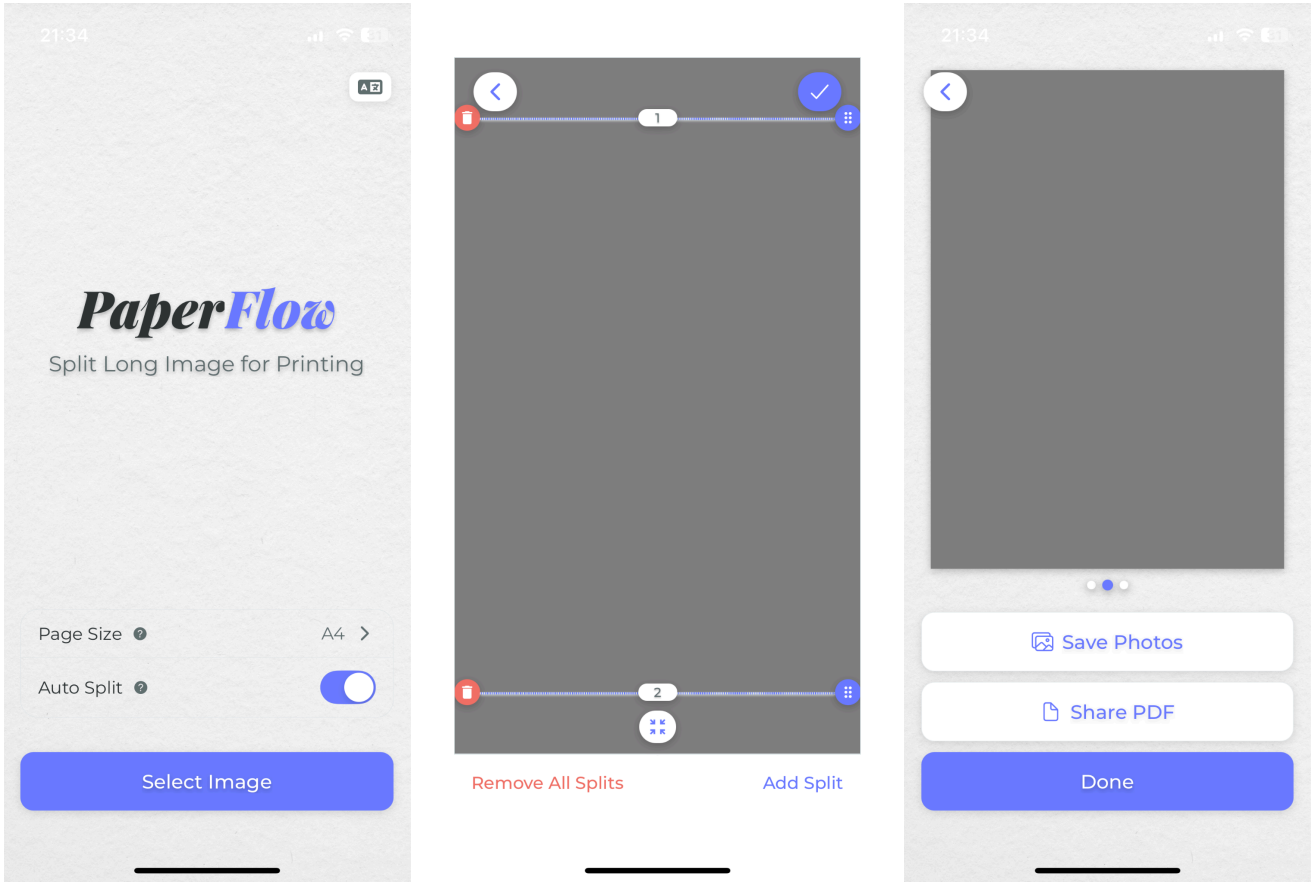
- Added a **background image** to evoke the idea of "paper".
- Introduced a **language toggle** button for switching between Chinese and English.
- Adjusted the **UI layout** for a better user experience.

3. **Split Screen**

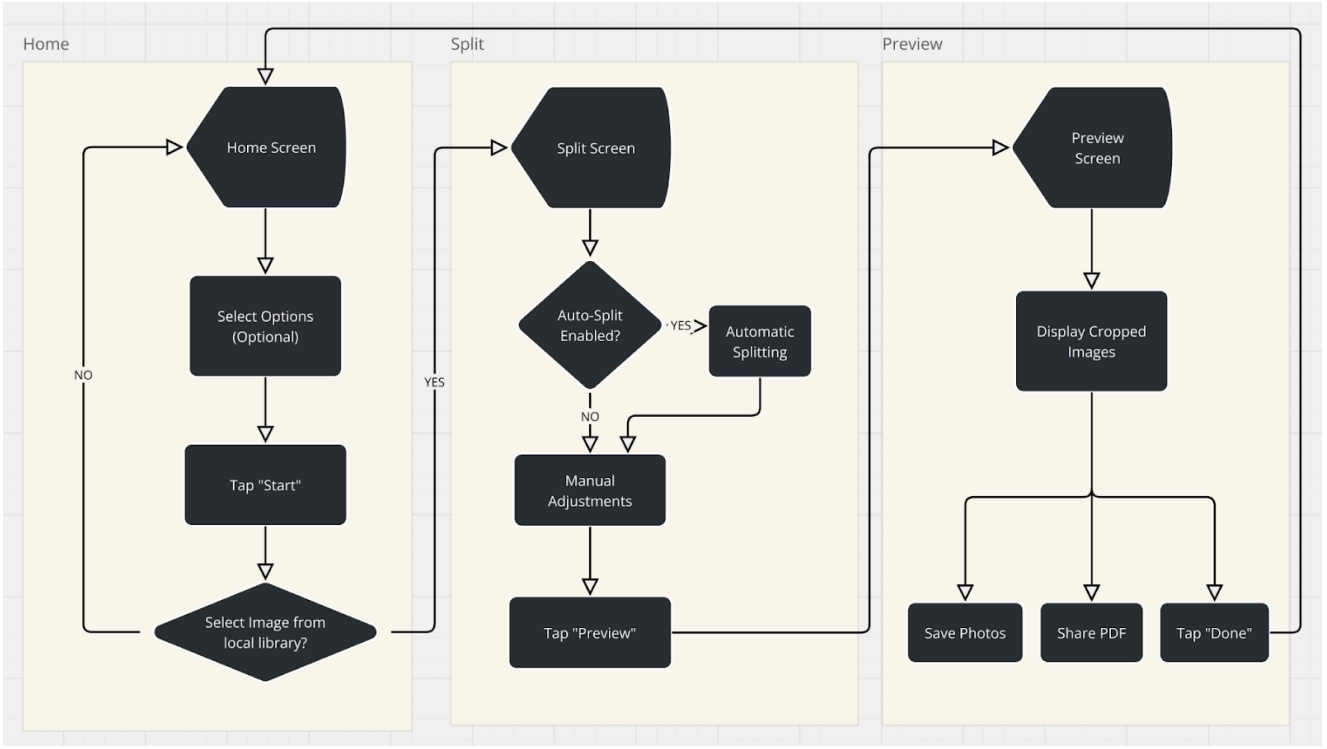
- Enabled image **zoom-in and zoom-out functionality**.
- Changed the **gesture indicator** from text to an icon button to reduce text burden.
- Changed the **"Back" and "Preview" buttons** from text to icons for a similar reason.
- Replaced the **loading indicator** with a spinner animation (which cannot be seen in the screenshot) for a similar reason.
- Minimised the **page number indicator** and placed it in the middle of the split line to avoid obstructing the image.

4. **Preview Screen**

- Replaced the grid layout with a **carousel** for result previews, allowing users to view larger images and check for cropping errors without extra click-to-zoom.
- Changed the **"Back" button** from text to an icon for consistency with the split screen.
- Removed the less-wanted **"Save PDF to File" and "Share Photos" functionalities** to reduce visual and functional clutter.
- Added **icons** to buttons for better non-text indication.



Wireframe Version 3



User Flow

The final version of the wireframes received very positive feedback. The interviewees found them clear and easy to follow, and the user interface was visually appealing and intuitive. This version of the wireframes provided a solid foundation for the app's development.

3. User Feedback

"I gathered user feedback through informal interviews and surveys with target users. Since I work in a business complex filled with law firms, I was able to easily connect with potential users—lawyers—who could test the app."

3.1 Early Concept Validation

Before committing to the app concept, I shared the idea for a crop-long-image-into-paper-size app with a small group of users to assess its necessity and appeal. Most participants expressed frustration with existing tools that either failed to preserve text integrity when splitting long images or were very time-consuming to use. They highlighted the importance of manual adjustments for fine-tuning split lines and emphasised the need for simplicity and efficiency in a user-friendly interface.

3.2 Wireframe Testing

Once the app concept was validated, I began developing wireframes and invited users to provide feedback on the wireframe design and its usability. This feedback was gathered through informal one-on-one interviews. The insights collected led to several vital adjustments, which I'll summarise below:

1. **OCR Text Detection and Privacy Concerns**

Initially, I intended to use an online OCR API to detect text positions and thus prevent truncation when splitting images. However, many users expressed concerns about data privacy and requested that the app function fully offline. While I researched local OCR options, I found their performance to be unsatisfactory. After extensive experimentation, I determined that OCR was unnecessary for the tool's core function. Instead, users could manually adjust the split lines when they cross text, which proved to be a manageable solution. This change simplified the app and alleviated privacy concerns.

2. **Clarity of Functionality**

In the first round of testing, many users were confused by specific features. For example, some didn't understand what the app was supposed to do just by viewing the home page, and others weren't sure whether the "Reset" button would reset the current operation or all split lines. I added more text clarifications and visual indications to the UI components to address this. However, in the second round of iteration, I noticed that the added text created a reading burden. I switched to a more icon-driven design, retaining only the necessary text to maintain clarity while improving visual simplicity.

3. **Avoiding UI Clutter**

Another significant insight came from observing how users interacted with the app's interface. In the first round, users struggled to process the five buttons on the Preview page. The layout was visually overwhelming, so in version 2 of the wireframe, I separated the save and share functions, moving the sharing options to the top-right corner to reduce clutter. However, users pointed out that sharing as a PDF is a commonly used function and shouldn't be nested. As a result, I simplified the UI further, removing the less desired "Share Photos" and "Save as PDF" features and leaving only the two most essential functions: "Save as Photos" and "Share as PDF."

4. **Reducing Unnecessary Operations**

During interviews, I asked test users to vocally walk me through their planned actions to better understand the user flow. Based on their feedback, I combined the "Image Upload" and "Start" actions into a single button to reduce an unnecessary click. Additionally, I changed the result preview from a grid layout to a carousel format, which allowed users to swipe through images without needing to click to enlarge each one. This simplified the navigation process and made the app feel more fluid.

3.3 Testing During Development

As mobile apps can perform differently across devices, I continued gathering feedback during development. I invited users with various devices to test the app and ensure a consistent user experience. This phase was critical for identifying and resolving issues specific to different platforms, especially Android, as I used an iOS

simulator for development. Much of the final stage of development was dedicated to debugging and addressing these device-specific issues. We'll discuss this further in the development section.

By actively incorporating user feedback at every stage, I ensured that PaperFlow not only addresses the core problem of splitting long images into paper-size pieces but also provides a seamless and enjoyable user experience. The overall user satisfaction rate rose from 5/10 to 9/10. Through iterative testing and refining based on real user input, the app evolved to meet user needs and preferences, resulting in a more intuitive and effective tool.

4. Prototyping

4.1 Technical Feasibility Validation

After validating the app concept, I immediately assessed the technical feasibility of my image-splitting app. Since the app's core feature is to efficiently split long text-based images into page sizes without losing text integrity, and my primary audience is Chinese users, I focused on selecting the right OCR tool for the auto-split job.

1. **Online OCR Testing: Baidu OCR API**

I began by testing the Baidu OCR API, the leading OCR service in the Chinese market.

Pros:

- User-friendly API interface
- Fast processing
- High accuracy

Cons:

- Paid service
- Cannot process long images directly—images must first be cropped into shorter segments, which require multiple API calls. This means manually combining the results, making it less efficient.

2. **Offline OCR Testing: Paddle OCR**

After receiving user feedback that rejected online services in favour of offline functionality, I turned to local OCR options. I tested Paddle OCR, known for its strong support of Chinese text.

Pros:

- Free
- Lightweight
- Can be deployed entirely offline
- Supports long-image text detection

Cons:

- Fatal design flaw: Paddle OCR is a combination of three core functionalities: text detection, text direction, and text recognition. Ideally, text detection should be independent of recognition, but in practice, when recognition is disabled, text detection becomes inaccurate. Enabling recognition improves accuracy, but the process becomes extremely slow, with text recognition consuming most of the processing time.
- The processing time for a 16-page-long image was 30 seconds, significantly reducing the app's usability. After thoroughly investigating the documentation, I couldn't find a solution, so I abandoned this approach.

3. **Shift to Non-OCR Solution**

With both online and offline OCR options ruled out, I shifted the focus away from text detection. Instead, I concentrated on splitting images based solely on page size. Users could manually adjust the splits later to avoid text truncation.

This approach was technically feasible. To split an image based on page size without OCR detection, I simply needed the aspect ratios of both the original image and the target page size. Then, I could use basic calculations to create the split lines.

4.2 Initial Prototype

The initial prototype of PaperFlow focused on implementing the core functionalities: image import, automatic splitting based on page size, and basic manual split line adjustments. Built using React Native and Expo, this version served as a proof of concept to demonstrate the viability of the proposed solution.

Key components of the initial prototype included:

- **Image Import**

Users could upload images from their local photo library.

- **Automatic Splitting**

Logic was implemented to calculate split line positions based on the chosen page size.

- **Manual Adjustments**

Essential gesture support allowed users to add, move, or delete split lines using click or press-and-drag gestures.

- **Image Processing**

The app manipulates the original image by cropping it into multiple segments based on the split line positions.

4.3 Iterative Refinements

After testing the initial prototype, I identified several areas for improvement and iterated on the design and functionality.

- **Improved UI/UX**

Refined UI components based on the revised wireframe. Redesigned the split interface to display delete and drag buttons and the page number on each split line.

- **Enhanced Gestures**

Expanded gesture support to include pinch-to-zoom and swipe-to-scroll, making manual adjustments smoother and more intuitive. I also added a zoom-in/zoom-out button as an alternative method to zoom in/out.

- **Result Preview**

Implemented a preview interface that displays cropped images in a carousel, allowing users to preview the final result.

- **Image Export**

Added functionality to export final images to the photo library or share them as a PDF.

4.4 Final Prototype

The final prototype incorporated all refinements and provided a complete version of the app. It added the following features to improve the user experience:

- **Translation**

Added translation support for both English- and Chinese-speaking users.

- **Local Storage**

Implemented local storage to save user preferences, including language choice, page size, and auto-split settings.

- **Animation**

Added an entrance animation to the app name and description, enhancing visual appeal. Also included an animated loading indicator during image processing to inform users that the app is working.

- **Haptic Feedback**

Implemented subtle haptic feedback when interacting with split lines to provide immediate visual feedback.

- **Alerts**

Alerts were added to notify users when an error occurs or when images are saved to the library.

This version was thoroughly tested by users and served as a solid foundation for further development.

5. Development

5.1 Project Structure

I set up the directory structure at the beginning of the development process to ensure modularity and maintainability. The following directories were created:

- **Types**: Defines custom TypeScript types.
- **Constants**: Stores global constants, such as the colour palette.
- **Components**: Contains reusable UI components.
- **Hooks**: Contains stateful logic functions.

- **Utils**: Contains stateless utility functions that perform specific operations.
- **Services**: Manages external service interactions, such as local storage.
- **Translations**: Stores UI copy for multiple languages.
- **Assets**: Contains fonts and images.
- **App**: Serves as the assembly shop for everything above.

This structured approach kept the codebase organised, making it easier to maintain, test, and extend with new features without disrupting existing functionality.

5.2 Tech Stack

In addition to the core React, React Native, and Expo libraries, I used the following libraries to enhance functionality:

- **React Native Gesture Handler**: Enables pinch, scroll, and pan gestures.
- **Expo Image Picker, Expo Image, Expo Image Manipulator, Expo Media Library**: Manages image import, display, manipulation, and export, respectively.
- **Expo Print, Expo Sharing**: Converts images to PDF and shares the converted PDF file.
- **React Native Swiper**: Provides carousel functionality for image previews.
- **React Native Walkthrough Tooltip**: Displays tooltips for user guidance.
- **React Native Async Storage**: Stores user preferences (e.g., language choice, page size) locally.
- **i18next, React i18next, Expo Localization**: Implements multi-language support.
- **React Native Reanimated**: Enhances animations.
- **Expo Haptics**: Provides haptic feedback for interactive elements.
- **React Native Safe Area Context**: Manages safe areas on iOS devices.
- **Expo Vector Icons**: Supplies icons for UI components.
- **Jest, React Native Testing Library**: Facilitates unit and integration testing.

This tech stack streamlined development and used existing solutions to ensure stability and performance while minimising redundant coding.

5.3 Technical Challenges

1. **Managing the Split Screen Complexity**

The split screen contained the app's core functionality, including:

- Displaying an image with pinch and scroll gestures.
- Rendering and managing split lines (add, move, delete) on the image.
- Cropping the image based on split line positions.

Initially, all logic and UI components were written in a single file, making debugging very difficult. There were gesture conflicts between the image and split lines. It was also messy to calculate and update split line positions (an array of y-positions on the original image) based on container dimensions, original image dimensions, display size, and page size.

To resolve these issues, I refactored the code:

- Moved image display logic to a *useZoomAndScroll* hook and *calculateImageMetrics* utility.
- Separated the split line logic into *useSplitManagement* hook and its UI component into *SplitLine.tsx*.
- Extracted image processing logic into a *useImageProcessing* hook.

This refactoring process, though time-consuming, significantly improved code organisation and maintainability. Debugging became easier, and users could now seamlessly zoom, scroll, and adjust split lines without errors. The index numbers on split lines also update correctly when modified.

2. **Converting Images to PDF**

I initially attempted to convert images directly into a single PDF file and ensure each page matched the exact image size. It didn't work. After extensive research, I realised that since the app's purpose was to split long images for *printing*, PDF pages should have a uniform size matching the selected paper dimensions.

The best solution was Expo Print, which allows images to be converted into HTML and then printed as a PDF. CSS styles controlled image placement to ensure:

- Each image occupied exactly one page.
- There were no unwanted margins or padding.
- Images were not cropped or split across multiple pages.

This approach guaranteed a seamless printing experience for users.

3. **Android-Specific Issues**

Developing primarily on an iOS simulator, I later discovered several Android-specific issues during user testing that required fixes:

- **UI Positioning**

Absolutely positioned UI components appeared too low on Android because Android does not have Dynamic Island "bangs". I introduced a *MARGINS* constant and applied platform-specific margins.

- **Font Rendering**

Custom fonts were not displayed on Android due to stricter naming conventions. The solution was to specify *fontFamily* correctly (e.g., *Montserrat-Medium* instead of *Montserrat* with *fontWeight: 500*).

- **Image Import Issues**

On Android, images did not appear in the split screen because Android could not locate the URI directly generated by Expo Image Picker. The fix was to copy the file to a persistent location and use the new URI on Android.

- **Blurry Image Display**

Android devices with high pixel densities displayed blurry image on the split screen due to the difference between logical pixels (DPI-independent) and physical pixels (device-specific). Using *PixelRatio*, I scaled the image up before scaling it down for display to ensure crisp rendering on high-resolution screens.

These solutions ensured that the app functioned smoothly across both iOS and Android, providing a consistent user experience regardless of the platform.

6. Unit Testing

The testing in this project follows Jest and React Native Testing Library best practices. Test files are located in the __tests__ subdirectories within each directory. For example, the test file for */components/ZoomControl.tsx* is located in */components/__tests__/ZoomControl-test.tsx*.

Each file in the app, components, services, hooks, and utils directories has a corresponding test file. Each test follows a structured approach: first, it mocks dependencies; then, it renders the component; and finally, it asserts the expected output. The tests cover snapshots, positive, negative, and edge cases.

During testing, I discovered that some edge cases were not properly handled. For instance, when a split line appeared at the bottom of an image, the last cropped section had a height of 0. To address this, I filtered out split lines before cropping. This demonstrates how testing contributes to the app's stability and reliability.

Additionally, I used test coverage reports to ensure thorough testing. By the end of the process, I wrote 149 tests in 29 test suites that covered 98.46% of the code, and all tests passed successfully.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	98.19	87.12	95.91	98.46	
app	93.9	68.18	86.36	93.75	
_layout.tsx	75	50	100	75	30,40
index.tsx	96.77	50	85.71	96.55	86
preview.tsx	100	100	100	100	
split.tsx	90.9	100	75	90.9	147,173
components	98.9	94.59	97.05	100	
AnimatedTitle.tsx	93.75	100	75	100	
AutoSplitOption.tsx	100	100	100	100	
BackArrow.tsx	100	100	100	100	
CheckArrow.tsx	100	100	100	100	
CustomButton.tsx	100	100	100	100	
ImageSwiper.tsx	100	100	100	100	
InfoTooltip.tsx	100	100	100	100	
LanguageOption.tsx	100	100	100	100	
LoadingIndicator.tsx	100	100	100	100	
PageSizeModal.tsx	100	100	100	100	
PageSizeOption.tsx	100	100	100	100	
SplitActions.tsx	100	100	100	100	
SplitLine.tsx	100	83.33	100	100	114
Text.tsx	100	50	100	100	6
ZoomControl.tsx	100	100	100	100	
constants	100	50	100	100	
Colors.ts	100	100	100	100	
Margins.ts	100	50	100	100	5-7
PageSizes.ts	100	100	100	100	
hooks	100	92.85	100	100	
useImageProcessing.ts	100	100	100	100	
useMediaLibrary.ts	100	100	100	100	
useSplitManagement.ts	100	100	100	100	
useZoomAndScroll.ts	100	83.33	100	100	43
services	100	100	100	100	
storage.ts	100	100	100	100	
translation.ts	100	100	100	100	
translations	0	0	0	0	
en.ts	0	0	0	0	
zh.ts	0	0	0	0	
types	0	0	0	0	
ImageDimensions.ts	0	0	0	0	
PageSize.ts	0	0	0	0	
utils	100	93.75	100	100	
calculateImageDisplay.tsx	100	100	100	100	
calculateImageMetrics.ts	100	100	100	100	
generatePdfFromImages.ts	100	100	100	100	
splitImage.ts	100	91.66	100	100	113

```
Test Suites: 29 passed, 29 total
Tests:       149 passed, 149 total
Snapshots:   5 passed, 5 total
Time:        1.169 s
Ran all test suites.
```

Test Coverage Report

Thorough testing not only ensures code correctness but also improves the application's stability and performance.

7. Evaluation

7.1 Functional Evaluation

PaperFlow was designed to address the challenge of splitting long-form text-based images into printable formats while ensuring text integrity and user convenience. The current app successfully meets its intended goals and solves the identified problem.

- **Meeting Intended Goals**

The app allows users to import a long image, automatically split it into standard paper sizes (A4, Letter, Legal), and manually adjust split lines to prevent text truncation. These features align with the project's primary objective of simplifying the process of preparing long images for printing. Additionally, the export functionality enables users to save results as individual images or a single PDF file, catering to diverse user needs.

- **Comparison Against Initial Expectations**

During the conceptual phase, the project aimed to create a lightweight, user-friendly tool that eliminated the inefficiencies of existing solutions, such as manual cropping or rigid equal-part splitting. The final product exceeds these expectations by offering intuitive manual adjustments, gesture-based interactions, and offline functionality. The removal of OCR-based text detection, initially considered but later deemed unnecessary, streamlined the app’s core functionality and addressed user privacy concerns.

- **Requirement Fulfillment**

All core requirements outlined in the project overview—image import, automatic splitting, manual adjustments, preview interface, and export functionality—are fully implemented. Furthermore, additional features like translation support, local storage, and haptic feedback enhance the app’s usability and accessibility, going beyond the initial scope.

In conclusion, PaperFlow not only fulfils its functional objectives but also introduces refinements that make it a robust and reliable solution for its target audience.

7.2 User Experience Evaluation

PaperFlow's user experience has been crafted to ensure visual appeal, ease of use, and overall satisfaction. Feedback from post-development user testing highlights the app's success in delivering an intuitive and enjoyable experience.

- **Visual Design and UI/UX**

The app's visual design is clean and professional, with a consistent colour palette and font pairing that evoke the theme of "paper." Using icons over excessive text reduces cognitive load, while interactive elements like tooltips and animated loading indicators enhance engagement. The carousel preview interface, introduced in the final wireframe iteration, allows users to view larger images and check for cropping errors seamlessly, improving the overall flow.

- **User-Friendliness**

Users praised the app's simplicity and efficiency, particularly the ability to combine image upload and start actions into a single button. Gesture-based functionalities, such as pinch-to-zoom and swipe-to-scroll, were well-received for their intuitiveness. Manual adjustments to split lines are straightforward, and the inclusion of haptic feedback provides immediate confirmation of user actions, reinforcing confidence in the app's responsiveness.

- **Post-Development Feedback**

Informal interviews and surveys conducted after development revealed a significant improvement in user satisfaction, rising from an initial rating of 5/10 to 9/10. Users appreciated the app's focus on reducing unnecessary operations and clutter, citing the streamlined preview and export pages as standout improvements.

Overall, the app's thoughtful design and iterative refinement based on user feedback have resulted in a highly user-friendly experience catering to non-tech users.

7.3 Innovation and Creativity

PaperFlow incorporates skills and concepts learnt throughout Topics 1–10 of the module, showcasing innovation and creativity in several key areas.

- **Application of Skills Learned**

The project used all the concepts learnt throughout the module, including wireframing, user flow diagrams, prototyping, iterative testing, and unit testing. It not only covers techniques learnt in the module – such as animation, gesture, alert, local data storage, haptics and vibration – but also dives into the untouched fields – such as multi-language support, image manipulation, media library, file format conversion, and file sharing.

- **Originality and Creative Elements**

PaperFlow stands out through its innovative approach to solving a niche yet widespread problem. Unlike existing image-splitting or general photo-editing tools that either truncate text or require extensive manual effort, PaperFlow combines automation with user control, offering a targeted, balanced, lightweight solution. Features like gesture-based adjustments, animated transitions, and multi-language support add creative touches that enhance the app's uniqueness. Moreover, the app's focus on privacy by functioning entirely offline sets it apart from other online tools in a market where data security is increasingly valued.

In summary, PaperFlow exemplifies innovation by solving a real-world niche problem with a targeted, lightweight, and privacy-focused solution. It not only meets the needs of its target audience but also provides a user-friendly, efficient, and visually appealing experience.