Name: Megan Brandreth
ID: 100778693
Class: Mobile Development Fall 2024
Date: December 2, 2024

Life Palette Lifestyle App
Individual Contribution Report

Individual Contribution

Notifications	- Implemented all notification framework - Implemented two individual options for user notifications, one daily notification and one weekly notification, as is appropriate for the nature of the application - Accounted for various use cases regarding the notifications implemented. - Allowed for both the in-app enabling and disabling of notifications, as well as customized re-enabling. - Ensured functional link between the notifications page within settings and the pages where notifications were implemented. - Implemented snackbars to inform the user of what notifications have been enabled or disabled after selection - Snackbars were also implemented for disabling notifications outside of the notifications page
Settings page & settings sub-pages	- Created and designed the main settings page through various iterations, as well as designed and implemented the main structure of all settings sub-pages - Ensured visually pleasing and standardized layout of all settings (and related) pages. - Linked settings page to the main page, ensured proper connection between all settings functions and their interactions with other pages throughout the application.
Stylization and automated theming	- Learned and implemented Flex Colour

	Scheme package - Created custom colour theme for both light and dark mode individually - Automated and implemented custom theming app-wide - Uploaded and handled font implementation across the app, including various stylization options - Ensured standardized colours and structure app-wide
Light/Dark mode	- Used custom FlexColourScheme and AdaptiveTheme to create a functional light and dark mode that applies across the entire application automatically - Used a single button push indicating light or dark mode to select preferred setting - Learned and became familiar with FlexColourScheme and the AdaptiveTheme package, utilizing both for effective implementation
Permission handling	- Learned about the nature of permission handling in Android apps and implemented appropriate measures to allow users to navigate their permissions for the application and system. - Handled notification permissions with regard to various use cases and user preferences, accounting for changing wants and needs of the user in regards to their notifications. - Implemented structure for easy management of cloud data deletion, worked concurrently with fellow teammates to ensure its proper and effective implementation in settings.
Code Formatting	- Implemented Flutter standard for formatting code and maintaining a professional and singular style across all files. - Made appropriate explanatory comments where necessary - Ensured consistent and appropriate use of various text styles and fonts across the app

Troubleshooting	- Helped groupmates to determine solutions to various issues (i.e theme automation not working as expected, customizing various widgets) - Assisted in implementation of Flex Colour Scheme automated theming through classmate files - Provided resources when necessary for new or unfamiliar packages to ensure group mates were informed and aware of implementations and related documentation

Notifications:

Daily and weekly notifications were implemented in the main journal page (daily) and the main recipe book page (weekly). Notifications were not considered necessary in other areas of the app. Notifications were implemented with code formatting standards in mind, as well as app-wide automatic theming with Flex Colour Scheme. When either button (located in the top right corner of the aforementioned pages) is clicked and notifications have not been set, the user is prompted to confirm the enabling of the notification. If notifications have been already set, the user is instead told that the notification already has been set, but is given the option to disable them. When users enter either the journal page, the recipe book page, or the notifications page in settings they are prompted to allow or deny notifications. Users will also be prompted at the time to enable alarms, which is necessary for the notifications to be functional due to android limitations. In the case that notifications have been denied and the user wishes to change this, the user has the option to select a button which will redirect them to the phone's settings page for the app where permissions for notifications are located. Due to limitations with Android, certain permissions are only changeable in system settings and not through the app interface. This is why we redirect the user specifically to the page for the app permissions within settings. Within the notifications page there is a switch which allows users to completely enable or disable notifications within the app; when disabled and then enabled afterwards, users are prompted with what notifications they want to enable, with the options of weekly only, daily only, both, or to cancel the process. Furthermore, once a user makes their selection, I've implemented snackbars to inform the user of the process' success. Snackbars were a later implementation that also made their way into the journal and recipe book pages when the user chooses to disable notifications there. There was a lot of difficulty with implementing notifications and making them functional – android requires special permissions to enable scheduled notifications, and figuring out how to consistently and accurately have those permissions granted was a struggle. Lots of time was spent tweaking and testing it until it worked as expected.

Settings page and Settings sub-pages

The settings page and its subpages went through various iterations. Initially there existed a search bar (SearchBar) intended to allow for the search (and subsequent redirect) of specific settings functionalities. Due to the simplicity of the settings, this was deemed unnecessary for the current iteration of the project. Furthermore, initially on the main page there were buttons for account deletion and logging out. As a team we decided that accounts were unnecessary due to the non-social nature of the application, and instead opted for the deletion of cloud data replacing account deletion, and that functionality moving into the account page. In its final iteration, the settings page redirects from the main page of the application and the user sees various settings page options. Each page redirect has some functionality, aside from the About page which introduces the user to the team and the purpose of the applications creation.

Stylization and Automated Theming

Stylization and complete automated theming across the entire app was achieved with FlexColorScheme (https://docs.flexcolorscheme.com/). FlexColorScheme is a package that allows me to create custom colours that would be automatically applied across the entire app, as well as specifying specific colours for a light/dark mode which will be explained later. The package was really interesting to discover and dive into – it wasn't covered within class, but it makes standardization across an app streamlined and ensures pages look unified by reducing potential errors page-by-page when the app theme is unclear and colour's used are not completely explained. Implementation of FlexColorScheme eliminates the need for tedious implementation of colour for each page, instead doing it automatically utilizing ThemeData. In FlexColorScheme, main, common widgets and items are specified, but there is room for further customization regarding background colours, specific item changes, etc. that make it an incredibly intuitive experience and well worth implementing. In order to assist fellow group members in understanding the implementation, its use, and expectations for them (i.e. only overriding code colours when necessary, automation meaning no need to implement colours themselves), I made sure to review group member code, and acted as the main assist in solving issues with integration in separate or added parts. There were a few instances where the theme did not apply as expected – to solve this, I worked with my group members to further tweak what is covered by FlexColorScheme. This was actually how I discovered further customization provided by the application (https://github.com/rydmike/flex_color_scheme/blob/master/INSIDE.md), and through this I narrowed down why certain aspects were not having the theme properly applied, and then what I could do to fix that. In the initial stages of implementation I had not been aware that the background colours were able to be changed, and was under the impression that they were stuck being a pure white and a pure black as was also the case with AdaptiveTheme. Fortunately, I discovered that the background colour of scaffolds, as well as general surface colour was alterable to be custom as well.

Light/Dark mode

Light and dark mode was implemented first with the FlexColorScheme theming and AdaptiveTheme. AdaptiveTheme (https://github.com/birjuvachhani/adaptive_theme) was created to allow for simple implementation of light/dark mode in applications. It made implementing my custom themes from FlexColorScheme possible, despite challenges I faced attempting to get it to work. A big issue I initially ran into was an attempt to have the app recognize the settings for light and dark mode on the system and automatically apply that to the app. AdaptiveTheme actually has 3 defaults, a default light mode, a default dark mode, and a system mode. With my original implementation, I hadn't realized that AdaptiveTheme's 'system' mode would read the system and then use its own defaults instead of the ones I'd specified in the code. Furthermore, It seemed that system mode ended up overriding light and dark mode selection completely. This error was due to two things. First of all, it happened because I had specified the adaptive system theme as the ThemeMode, assuming that doing so would allow the system to understand it was the initial state of the app: for many obvious reasons, this explains why the light and dark modes were overridden and why they weren't functional. Secondly, even after solving that issue, I hadn't been able to determine what I could put for ThemeData instead that would make it change. After some consideration, I determined that using a ternary operator would fit best for the situation. I knew I needed to determine what was the current system theme, but I also needed that to determine what the current chosen theme was and be able to change that like a switch as well. Essentially, the current implementation sets the initial theme based on my custom light and dark mode. Then, the ternary operator does a check based on the current theme by checking if the theme is light mode. If the theme is light mode, and we are attempting to change to light mode, it maintains its visual state. If the theme isn't it changes to our custom specified dark mode. All of this is handled in the main, and it's all very compact. It was an incredibly useful tool, even though it was one that took me a considerable amount of time to fully understand.