[Padawan Wallet]

Summer of Bitcoin 2023

Advanced Bitcoin features on Padawan Wallet

Name and Contact Information

Name: Prakhar Agarwal

Github username: Prakhar-Agarwal-byte

Discord username: @Prakhar#9227

Email: prakharagarwal3031@gmail.com

Alternative Mail: prakhar.agarwal.mec20@itbhu.ac.in

University info

- **University Name:** Indian Institute of Technology (BHU) Varanasi
- **Major:** Mechanical Engineering
- **Current year and expected graduation date:** 3rd year (junior). Expected graduation 30th April 2024

Degree: B.Tech. (Bachelor of Technology (4 years))

Website: prakharcodes.tech

LinkedIn: prakhar-agarwal-byte

Country: India

Time Zone: IST (UTC+05:30)

Table of Contents

Title

Advanced Bitcoin features on Padawan Wallet

Synopsis

The Padawan Wallet is an open-source Android wallet designed to help newcomers learn about Bitcoin by using testnet. It requires more polishing, a robust test suite, code review, and additional features. The project aims to provide a strong tool for learning and practicing Bitcoin with Padawan. I will add features to the application over the summer, expand its functionalities, and increase the wallet's coverage of the typical mobile wallet workflows. The expected outcome is a well-tested Padawan Wallet, covering a wider range of functionalities for typical mobile wallet users. The project also involves adding features to the faucet used in Padawan to build its resilience.

Project Plan

This project aims to release a strong, simple, but complete feature set for version 1.0. I have proposed a few ideas with code snippets to demonstrate its rough implementation. I will also complete all the issues of the v1.0.0 milestone to make the app release-ready, some of whose implementations I have already provided.

Add an introduction tour of the app on the first startup

An introduction tour is a great way to help new users get started with the app. It can highlight key features, provide tips and tricks, and explain how to perform basic functions. By providing this guidance upfront, we can ensure that users have a positive experience with the app and are more likely to continue using it.

I will follow these steps to add an introduction tour in PadawanWallet:

1. Trigger the introduction tour on the app's first launch.

2. Create a sequence of steps to guide the user through the introduction tour. This could include highlighting key features of the app, providing tips and tricks, and explaining how to perform basic functions.

3. Implement a visual cue to let the user know the introduction tour is available. This could be a button labeled "Get Started" or a pop-up that appears automatically.

4. Write code to display the introduction tour when the trigger conditions are met. This could involve creating a new screen or activity in the app specifically for the tour or using overlays or tooltips to highlight key areas of the app.

5. Allow the user to skip the tour if they prefer. Provide an option to skip the tour that is clearly visible and easy to access.

6. Store a flag or preference to indicate that the user has completed the introduction tour so they do not see it again on subsequent app launches.

In our implementation using Kotlin and Jetpack Compose, we created two composable functions. The WelcomeScreen function displays a welcome message and two buttons to start the tour and skip it. The IntroductionTour function would contain the actual code to display the tour, including overlays and tooltips. We also used a MutableState variable to track whether this is the app's first launch and to trigger the WelcomeScreen if it is.

```
@Composable
fun WelcomeScreen(onStartTour: () -> Unit, onSkipTour: () -> Unit)
{
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(
            text = "Welcome to Padawan Wallet",
            style = MaterialTheme.typography.h4,
            modifier = Modifier.padding(bottom = 16.dp)
        )
        Text(
            text = "Would you like to take a quick tour of the
app's features?",
            style = MaterialTheme.typography.body1,
            modifier = Modifier.padding(bottom = 32.dp)
        )
        Button(
            onClick = { onStartTour() },
            modifier = Modifier.fillMaxWidth()
        ) {
            Text(text = "Get Started")
        }
        TextButton(
            onClick = { onSkipTour() },
            modifier = Modifier
                .padding(top = 16.dp)
                .fillMaxWidth()
        ) {
            Text(text = "Skip")
        }
    }
}

@Composable
fun IntroductionTour(onCompleteTour: () -> Unit) {
    // your code to display the introduction tour, including
```

```
overlays and tooltips
    // once the user completes the tour, call onCompleteTour()
}

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContent {
            var isFirstLaunch by remember { mutableStateOf(true) }
            if (isFirstLaunch) {
                WelcomeScreen(
                    onStartTour = {
                        isFirstLaunch = false
                        IntroductionTour(onCompleteTour = {})
                    },
                    onSkipTour = {
                        isFirstLaunch = false
                    }
                )
            } else {
                // your code to display the main screen of Padawan
Wallet
            }
        }
    }
}
```

I will use the Balloon library for implementing the IntroductionTour composable. It is a lightweight library that allows us to create customizable and animated tooltips that can be used to highlight important features or information in your app.

```
@Composable
fun IntroductionTour() {
    val coroutineScope = rememberCoroutineScope()
    val balloonPadding = 16.dp
    val showBalloon = remember { mutableStateOf(false) }
    val anchorView = remember { mutableStateOf<Ref>(null) }
```

```
    val firstTimeUser = remember { mutableStateOf(false) }
    val balloonMessages = listOf(
        "Welcome to Padawan Wallet! To get started, tap on the +
button to add a new transaction.",
        "Great job! You've added your first transaction. Now you
can categorize your spending.",
        "Don't forget to set a budget for your expenses. Tap on
the icon at the bottom to access the budgeting feature.",
        "You're all set! Enjoy using Padawan Wallet to track your
expenses and budget.",
    )
    var currentMessageIndex by remember { mutableStateOf(0) }

    if (firstTimeUser.value) {
        LaunchedEffect(Unit) {
            coroutineScope.launch {
                delay(500) // Wait for layout to finish before
showing balloon
                showBalloon.value = true
            }
        }
    }

    Box(modifier = Modifier.fillMaxSize()) {
        // Your app content goes here

        if (showBalloon.value) {
            Balloon(
                modifier =
Modifier.align(Alignment.TopEnd).padding(end = balloonPadding),
                arrowSize = 10.dp,
                arrowOrientation = ArrowOrientation.Bottom,
                arrowPosition = 0.5f,
                backgroundColor = MaterialTheme.colors.primary,
                contentColor = Color.White,
                contentPadding = PaddingValues(balloonPadding),
                cornerRadius = 4.dp,
                showArrow = true,
                text = {
                    Text(
                        text =
```

```
balloonMessages[currentMessageIndex],
                        style = MaterialTheme.typography.body2,
                        modifier = Modifier.padding(vertical =
8.dp)
                    )
                },
                onCloseRequest = {
                    if (currentMessageIndex < balloonMessages.size
- 1) {
                        currentMessageIndex++
                        showBalloon.value = true
                    } else {
                        showBalloon.value = false
                    }
                }
            )
        }
    }
}
```

Overall, an introduction tour can be a valuable addition to any app, especially for first-time users. By providing a clear and easy-to-follow guide to the app's features and functionality, we can help users get up and running quickly and ensure a positive experience.

Disable touch opacity when Bottom sheet is expanded

We are currently not disabling the background touch opacity whenever the confirm transaction bottomsheet expands. This may lead to the Application crash. We can switch to the Modal bottom sheet, which by default, can disable the background touch opacity.

Generating an address displays a different string for a moment

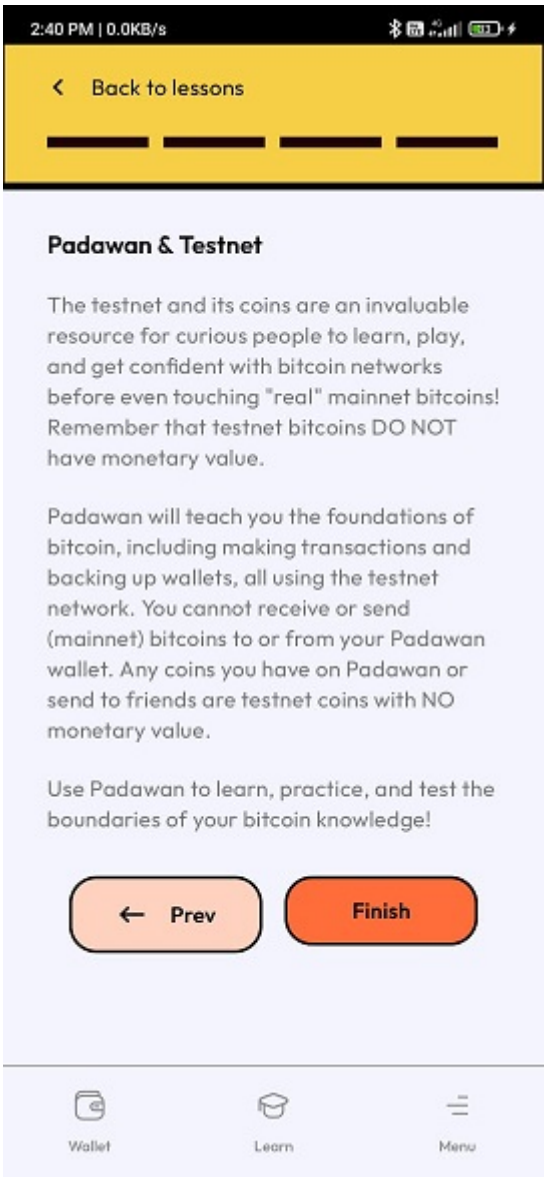
The QR image displays the string "No address yet" for a split second while the actual first address is computed. This can be fixed by removing the “No address yet” string from the WalletViewModel.kt From:

```
private var _address: MutableLiveData<String> =
MutableLiveData("No address yet")
```

To:

```
private var _address: MutableLiveData<String> =
MutableLiveData("")
```

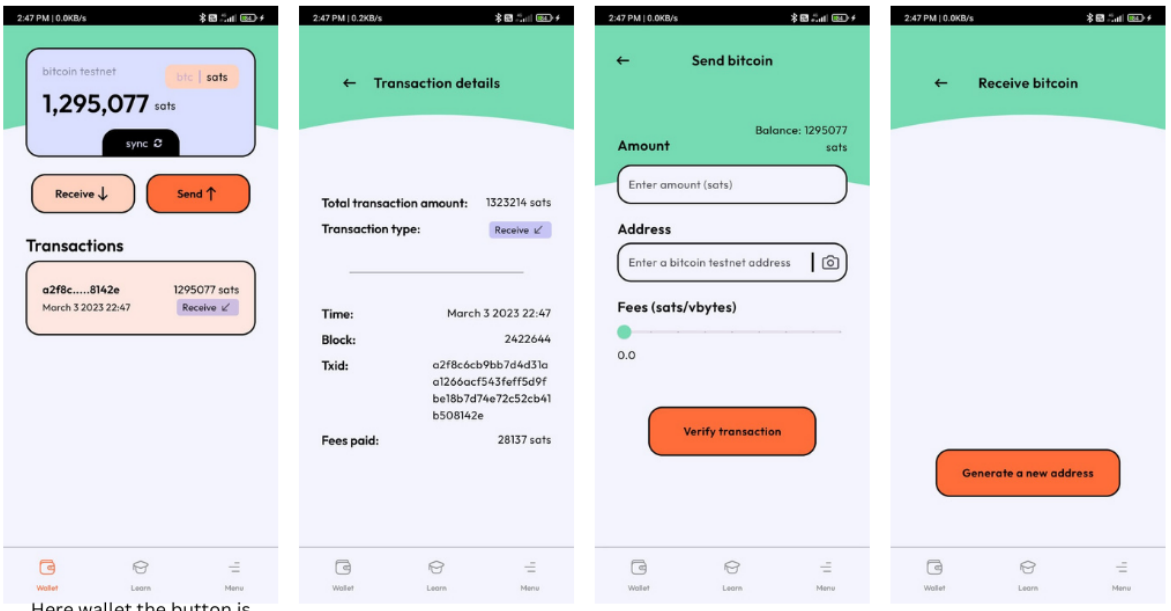
Make the "Finish" button of the same size in chapters



The Finish button here is smaller than the Prev and Next buttons.

The two buttons sharing a row and filling up the height + using each the same width proportion inside the row will ensure they always just look the same.

Highlight the Bottom Navigation Tab



Here wallet the button is orange

Its deselected on opening other screens

The Wallet, Learn and Menu bottom navigation tab gets deselected when any screen inside of it is opened. The correct tab should remain selected. We can fix this by properly pushing the fragments in the backstack and later checking if the hierarchy of the current fragment contains one of those bottom nav tabs, if it matches then highlight the corresponding tab.

Need of progress bar while clicking "Create new address"

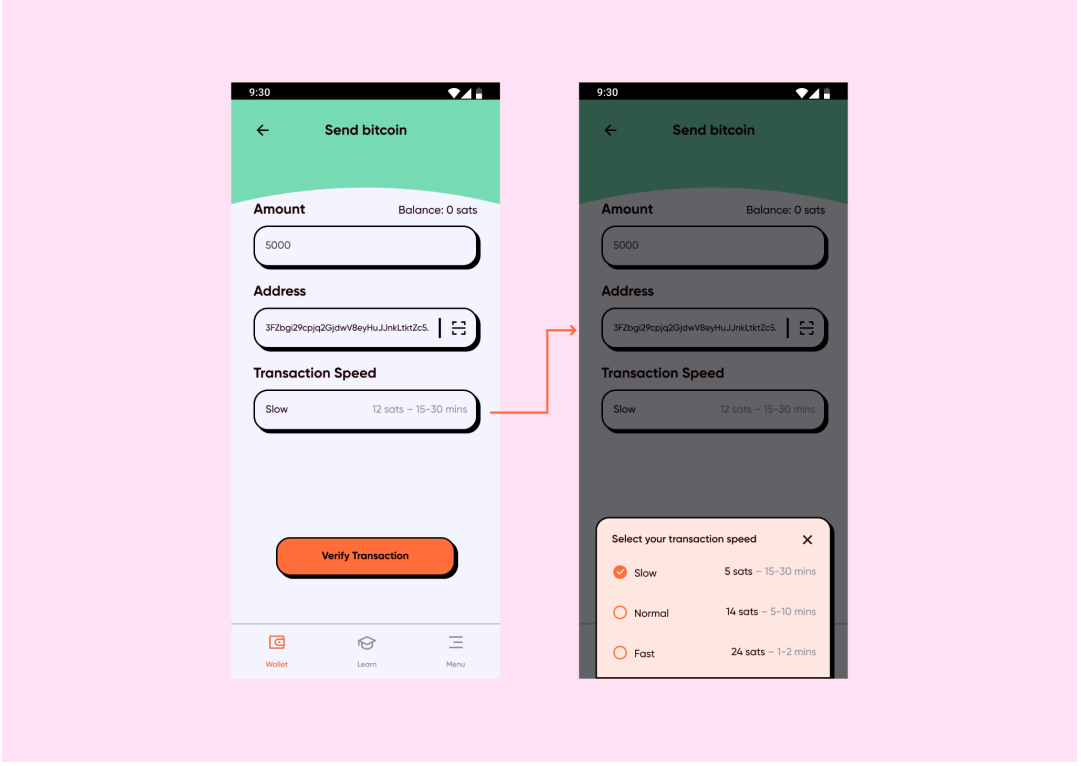
When we click on create new address button, there should be a progress bar to show the loading state when the QR is generated. We can achieve this by toggling the isGeneratingQRState to true or false.

```
isGeneratingQRState = true
viewModel.updateLastUnusedAddress()
isGeneratingQRState = false
```

Making Transaction Speed/Fees Clearer

As @Noyi-24-7 pointed out, some people using the fees slider don't know what the “fees” part on the sending bitcoin page is used for or whether it affects the speed of sending. They find it difficult to understand how hard it would work to send Bitcoin quickly.

> Instead of a slider, we can list the options of speeds, i.e., Fast, Normal, and Slow, and mention the fees associated with each type of transaction inside a modal.



▲ *Design by [@noyi-24-7]*

Rough Implementation of the feature

```
class WalletFragment : Fragment() {
    // other properties and methods here

    private var transactionSpeed: TransactionSpeed =
TransactionSpeed.FAST
    private val transactionFees = mutableMapOf(
        TransactionSpeed.FAST to 0.0025,
        TransactionSpeed.MEDIUM to 0.0015,
        TransactionSpeed.SLOW to 0.0005
    )

    private fun updateTransactionSpeed(speed: TransactionSpeed) {
        transactionSpeed = speed
    }

    private fun showTransactionSpeedBottomSheet() {
        val bottomSheet =
ModalBottomSheetLayout(LocalContext.current)

        bottomSheet.show {
            val state =
rememberModalBottomSheetState(ModalBottomSheetValue.Hidden)
            ModalBottomSheetLayout(modalBottomSheetState = state)
{
                Column(
                    modifier = Modifier.fillMaxWidth(),
                    horizontalAlignment =
Alignment.CenterHorizontally
                ) {
                    Text(text = "Transaction Speed")

                    Row(
                        modifier = Modifier.fillMaxWidth(),
                        horizontalArrangement =
Arrangement.SpaceBetween
                    ) {
                        Button(
                            onClick = {
```

```
updateTransactionSpeed(TransactionSpeed.FAST)
                                state.hide()
                            },
                            modifier = Modifier.weight(1f)
                        ) {
                            Text(text = "Fast")
                        }

                        Button(
                            onClick = {

updateTransactionSpeed(TransactionSpeed.MEDIUM)
                                state.hide()
                            },
                            modifier = Modifier.weight(1f)
                        ) {
                            Text(text = "Medium")
                        }

                        Button(
                            onClick = {

updateTransactionSpeed(TransactionSpeed.SLOW)
                                state.hide()
                            },
                            modifier = Modifier.weight(1f)
                        ) {
                            Text(text = "Slow")
                        }
                    }
                }
            }
        }
    }

    @Composable
    fun TransactionSpeedSelector() {
        Column(
            modifier = Modifier.fillMaxWidth(),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
```

```
            Text(text = "Transaction Speed")

            Row(
                modifier = Modifier.fillMaxWidth(),
                horizontalArrangement = Arrangement.SpaceBetween
            ) {
                TextButton(
                    onClick = { showTransactionSpeedBottomSheet()
},
                    modifier = Modifier.weight(1f)
                ) {
                    Text(text = transactionSpeed.name)
                }

                Text(
                    text = "Estimated fee:
${transactionFees[transactionSpeed]} BTC",
                    modifier = Modifier.weight(1f)
                )
            }
        }
    }

    @Composable
    fun WalletScreen() {
        // other composable functions here
        Column(
            modifier = Modifier.fillMaxSize(),
            verticalArrangement = Arrangement.Top,
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            // display wallet balance and other info here
            TransactionSpeedSelector()
        }
    }
}
```

In this code, we define a new function showTransactionSpeedBottomSheet that creates and displays a ModalBottomSheetLayout containing the buttons to select the

transaction speed. The updateTransactionSpeed function is called when a button is clicked to update the current transaction speed.

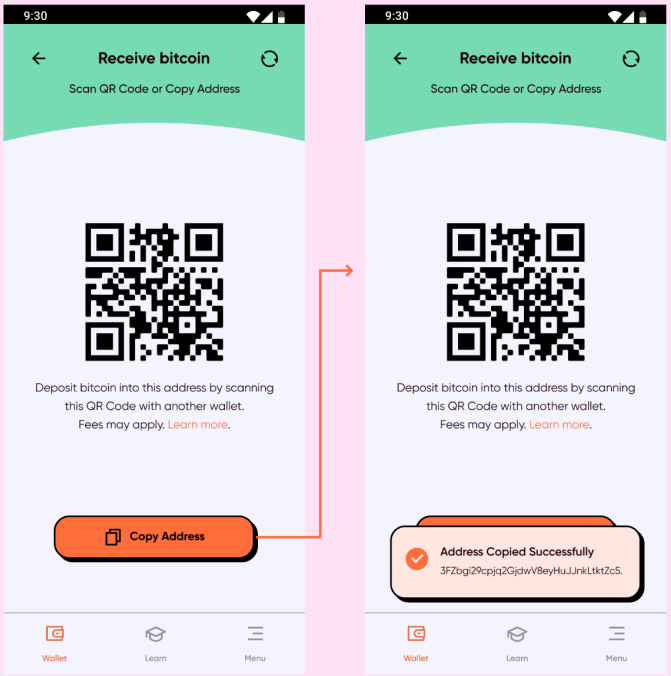
We also modify the TransactionSpeedSelector function to display the current transaction speed name in a TextButton and show the estimated transaction fee in a Text element next to it.

Finally, we add the TransactionSpeedSelector function to the WalletScreen composable function, which displays the wallet balance and other info, and returns a composable layout that includes the TransactionSpeedSelector.

With this implementation, users of Padawan Wallet can easily access the transaction speed selection options in a clean and simple bottom sheet, and can view the estimated transaction fee alongside their selection.

Making it easier for users to copy their bitcoin address

As of now when users want to receive bitcoin and want to copy their bitcoin address, it is bit confusing as to how to do that, or if you already know, it is a lot of steps, which include highlighting the address first then using the native copy button. Which is pretty difficult for users it seems.



▲ *Design by @noyi-24-7*

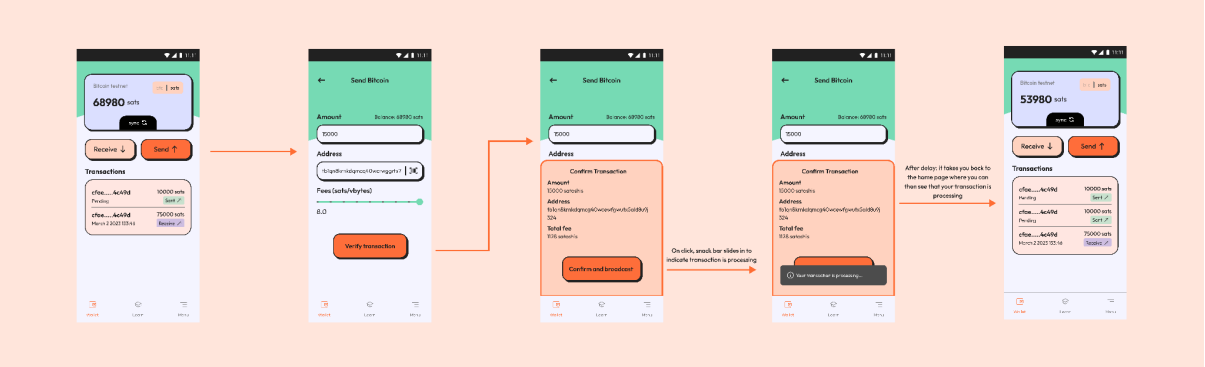
Code for copying the address on the clipboard

```
private fun copyToClipboard(text: String, label: String) {
        val clipboardManager = ContextCompat.getSystemService(
            requireContext(),
            ClipboardManager::class.java
        ) as ClipboardManager

clipboardManager.setPrimaryClip(ClipData.newPlainText(label,
text))
        Toast.makeText(
            requireContext(),
            "Copied $label to clipboard",
            Toast.LENGTH_SHORT
        ).show()
    }
```

Add feedback when user carry out transaction

When users carry out a testnet transaction, there's no feedback to let them know if their transaction went through or not unlike other Bitcoin wallets. We could add a feedback (snackbar that shows the transaction is processing just like we did for 'Reset Completed Chapters')
Here is what it should look like:



▲ *Design by @thetimileyin*

```
class WalletFragment : Fragment() {
    // other properties and methods here

    private fun showSnackbar(message: String) {
        Snackbar.make(requireView(), message,
Snackbar.LENGTH_SHORT).show()
    }

    @Composable
    fun SendButton() {
        // other code here
        Button(
            onClick = {
                // perform transaction here
                showSnackbar("Transaction successful!")
            },
            // other button properties here
```

```
        ) {
            Text(text = "Send")
        }
    }

    @Composable
    fun WalletScreen() {
        // other composable functions here
        Column(
            modifier = Modifier.fillMaxSize(),
            verticalArrangement = Arrangement.Top,
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            // display wallet balance and other info here
            SendButton()
        }
    }
}
```

Add transaction sorter to transaction list

One way to make it easier for users to view their transaction history is by adding a transaction sorter to the transaction list. This feature allows users to sort their transactions based on specific criteria, such as date or amount. In this example, we'll use Kotlin and Jetpack Compose to add a transaction sorter to the transaction list in Padawan Wallet.

First, we define an enum class SortType that represents the sorting criteria:

```
enum class SortType {
    DATE_ASCENDING, DATE_DESCENDING, AMOUNT_ASCENDING,
AMOUNT_DESCENDING
}
```

Next, we define a function sortTransactions that takes a list of transactions and a SortType parameter and returns the sorted list of transactions:

```
fun sortTransactions(transactions: List<Transaction>, sortType:
SortType): List<Transaction> {
    return when (sortType) {
        SortType.DATE_ASCENDING -> transactions.sortedBy { it.date
}
        SortType.DATE_DESCENDING ->
transactions.sortedByDescending { it.date }
        SortType.AMOUNT_ASCENDING -> transactions.sortedBy {
it.amount }
        SortType.AMOUNT_DESCENDING ->
transactions.sortedByDescending { it.amount }
    }
}
```

In the TransactionList composable, we use the remember function to cache the sorted transactions based on the sortType parameter. This avoids unnecessary re-sorting when the TransactionList composable is recomposed:

```
@Composable
fun TransactionList(transactions: List<Transaction>, sortType:
SortType) {
    val sortedTransactions = remember(transactions, sortType) {
        sortTransactions(transactions, sortType)
    }
```

```
    Column(Modifier.verticalScroll(rememberScrollState())) {
        // Add sort type selector
        SortTypeSelector(sortType = sortType)

        // Display sorted transaction list
        sortedTransactions.forEach { transaction ->
            TransactionItem(transaction = transaction)
        }
    }
}
```

In the SortTypeSelector composable, we create two buttons to represent the SortType options: "Date" and "Amount". When a button is clicked, we call the onSortTypeChange function to update the sortType parameter:

```
@Composable
fun SortTypeSelector(sortType: SortType) {
    Row(
        Modifier.fillMaxWidth().padding(vertical = 8.dp),
        horizontalArrangement = Arrangement.SpaceEvenly
    ) {
        SortTypeButton(
            text = "Date", selected = sortType ==
SortType.DATE_ASCENDING,
            onClick = {
                if (sortType == SortType.DATE_ASCENDING) {
                    // Sort descending if already sorted ascending
                    onSortTypeChange(SortType.DATE_DESCENDING)
                } else {
                    // Sort ascending if not already sorted
ascending
                    onSortTypeChange(SortType.DATE_ASCENDING)
                }
            }
        )
        SortTypeButton(
            text = "Amount", selected = sortType ==
SortType.AMOUNT_ASCENDING,
```

```
            onClick = {
                if (sortType == SortType.AMOUNT_ASCENDING) {
                    // Sort descending if already sorted ascending
                    onSortTypeChange(SortType.AMOUNT_DESCENDING)
                } else {
                    // Sort ascending if not already sorted
ascending
                    onSortTypeChange(SortType.AMOUNT_ASCENDING)
                }
            }
        )
    }
}
```

Finally, we define the SortTypeButton composable that represents a single sort type button. We use the selected parameter to set the button's background color and the onClick parameter to handle button clicks.

```
@Composable
fun SortTypeButton(text: String, selected: Boolean, onClick: () ->
Unit) {
    Button(
        onClick = onClick,
        colors = ButtonDefaults.buttonColors(
            backgroundColor = if (selected)
MaterialTheme.colors.primary else Color.Transparent,
            contentColor = if (selected) Color.White else
MaterialTheme.colors.primary
        ),
        shape = MaterialTheme.shapes.small
    ) {
        Text(text = text)
    }
}
```

With these composable functions defined, we can now use the TransactionList composable to display a list of transactions sorted by date or amount, depending on the user's preference.

To use the TransactionList composable in a screen, we can pass in a list of transactions and the current sortType parameter. For example, in the TransactionsScreen composable:

```
@Composable
fun TransactionsScreen(transactions: List<Transaction>) {
    var sortType by remember {
mutableStateOf(SortType.DATE_DESCENDING) }

    Column(Modifier.fillMaxSize()) {
        // Add transaction list
        TransactionList(transactions = transactions, sortType =
sortType) {
            sortType = it
        }
    }
}
```

Here, we initialize the sortType parameter to SortType.DATE_DESCENDING and pass it to the TransactionList composable. We also define a lambda expression that updates the sortType parameter when the user selects a new sort type.

In conclusion, adding a transaction sorter to the transaction list in Padawan Wallet can greatly improve the user experience by allowing users to easily view and organize their transaction history. By leveraging Kotlin and Jetpack Compose, we can create a flexible and reusable component that can be used in any screen that displays a transaction list.

Fix bugs related to smaller screens

I will fix most of the bugs due to smaller screen sizes. It's important to thoroughly test the app on different screen sizes and resolutions and identify any layout overlap, text truncation, or UI elements too small to be easily tapped. To address these issues, I will use responsive design principles and tools such as ConstraintLayout to create flexible and adaptive layouts that work well on smaller screens. I will also prioritize fixing critical issues, such as preventing the app from functioning properly or significantly impacting the user experience. By taking these steps, I can ensure that the app is optimized for various screen sizes and resolutions, providing a seamless user experience across devices.

Comprehensive testing of Padawan wallet

Comprehensive testing of the Padawan wallet is crucial to ensure that the app functions as expected and is free of bugs. Here is an example of how comprehensive testing can be achieved using JUnit and Espresso frameworks in Android:

```
import androidx.test.core.app.ApplicationProvider
import androidx.test.espresso.Espresso
import androidx.test.espresso.assertion.ViewAssertions
import androidx.test.espresso.matcher.ViewMatchers
import androidx.test.ext.junit.rules.ActivityScenarioRule
import org.junit.Rule
import org.junit.Test

class PadawanWalletTest {

    @get:Rule
    val activityRule =
```

```
ActivityScenarioRule(MainActivity::class.java)

    @Test
    fun testSendTransaction() {
        // Simulate user input for sending a transaction

Espresso.onView(ViewMatchers.withId(R.id.send_button)).perform(cli
ck())

        // Enter transaction details

Espresso.onView(ViewMatchers.withId(R.id.recipient_address_field))
.perform(typeText("recipient_address"))

Espresso.onView(ViewMatchers.withId(R.id.amount_field)).perform(ty
peText("1.0"))

        // Click on Send button

Espresso.onView(ViewMatchers.withId(R.id.send_transaction_button))
.perform(click())

        // Check if transaction was successful

Espresso.onView(ViewMatchers.withText(R.string.transaction_success
ful))

.check(ViewAssertions.matches(ViewMatchers.isDisplayed()))
    }

    @Test
    fun testReceiveTransaction() {
        // Simulate user input for receiving a transaction

Espresso.onView(ViewMatchers.withId(R.id.receive_button)).perform(
click())

        // Check if address is displayed

Espresso.onView(ViewMatchers.withId(R.id.address_textview))
```

```
.check(ViewAssertions.matches(ViewMatchers.isDisplayed()))
    }

    @Test
    fun testWalletBalance() {
        // Check if wallet balance is displayed

Espresso.onView(ViewMatchers.withId(R.id.balance_textview))

.check(ViewAssertions.matches(ViewMatchers.isDisplayed()))
    }

    @Test
    fun testSettingsScreen() {
        // Simulate user input for opening the settings screen

Espresso.onView(ViewMatchers.withId(R.id.settings_button)).perform
(click())

        // Check if settings screen is displayed

Espresso.onView(ViewMatchers.withText(R.string.settings_screen_tit
le))

.check(ViewAssertions.matches(ViewMatchers.isDisplayed()))
    }

    @Test
    fun testAboutScreen() {
        // Simulate user input for opening the about screen

Espresso.onView(ViewMatchers.withId(R.id.about_button)).perform(cl
ick())

        // Check if about screen is displayed

Espresso.onView(ViewMatchers.withText(R.string.about_screen_title)
)

.check(ViewAssertions.matches(ViewMatchers.isDisplayed()))
    }
```

```
    @Test
    fun testNavigationDrawer() {
        // Simulate user input for opening the navigation drawer

Espresso.onView(ViewMatchers.withContentDescription(R.string.navig
ation_drawer_content_desc)).perform(click())

        // Check if navigation drawer is displayed

Espresso.onView(ViewMatchers.withText(R.string.navigation_drawer_t
itle))

.check(ViewAssertions.matches(ViewMatchers.isDisplayed()))
    }
}
```

In this example, we simulate user input for various scenarios, such as sending and receiving transactions, opening the settings and about screens, and opening the navigation drawer. We then use the Espresso framework to perform assertions on the UI elements to ensure that the app behaves as expected. The tests are run using the JUnit framework, which provides a comprehensive test results report. By running comprehensive tests like these, developers can ensure that the Padawan wallet is reliable and bugs-free.

```
@RunWith(AndroidJUnit4::class)
class WalletTest {

    @Test
    fun testGeneratePrivateKey() {
        val wallet = Wallet()
        val privateKey = wallet.generatePrivateKey()
        assertNotNull(privateKey)
        assertEquals(64, privateKey.length)
    }

    @Test
    fun testGeneratePublicKey() {
        val wallet = Wallet()
        val privateKey = wallet.generatePrivateKey()
        val publicKey = wallet.generatePublicKey(privateKey)
```

```
        assertNotNull(publicKey)
        assertEquals(130, publicKey.length)
    }

    @Test
    fun testSignTransaction() {
        val wallet = Wallet()
        val privateKey = wallet.generatePrivateKey()
        val transaction = mapOf("sender" to "Alice", "recipient" to "Bob",
"amount" to 1)
        val signature = wallet.signTransaction(privateKey, transaction)
        assertNotNull(signature)
        assertTrue(signature.isNotEmpty())
    }

    @Test
    fun testVerifySignature() {
        val wallet = Wallet()
        val privateKey = wallet.generatePrivateKey()
        val publicKey = wallet.generatePublicKey(privateKey)
        val transaction = mapOf("sender" to "Alice", "recipient" to "Bob",
"amount" to 1)
        val signature = wallet.signTransaction(privateKey, transaction)
        assertTrue(wallet.verifySignature(publicKey, signature, transaction))
    }
}
```

Devkit Wallet

The Devkit wallet is a demo app for the bitcoin development kit Android language bindings, and a lot of developers use it to become familiar with the library. Some of the features I will add to this project are mentioned below:

1. Add multi-sig functionality
2. Add OP_RETURN functionality
3. Display address path when showing address

I will learn more about these over the summers and have discussions with my mentor to enable them on the app.

Project Timeline

Community Bonding and Onboarding(May 1, 2023 - May 15, 2023)	
`May 1, 2023 - May 15, 2023	• Get familiar with the codebase and the mentors. • Discuss my approaches, logic, and milestones with the mentor. • Attend webinars related to Bitcoin.
Project Period(May 15, 2023 - Aug 15, 2023)	
May 15, 2023 - May 31, 2023	• Setup local dev environment • Add introduction tour of the app • Create a doc of changes
May 31, 2023 - Jun 15, 2023	• Make transaction fees clearer • Make it easier to copy bitcoin address • Add feedback when user carry out transaction
Jun 15, 2023 - Jun 30, 2023	• Fix bugs related to smaller screens • Keep on documenting the changes • Write tests
Jun 30, 2023 - Jul 15, 2023	• Work on finishing the v1.0.0 milestone • Discuss any other features

	that we can add for this release.
Jul 15, 2023 - Jul 31, 2023	• Extensive testing, catching bugs, fixing them, and improving the code coverage.
Aug 1, 2023 - Aug 15, 2023	• Display address path when showing the addresses • Add multi-sig functionality • Add OP_RETURN functionality • Test the devkit wallet for these features

Future Deliverables

I intend to stay with the community after the Summer of Bitcoin. I will try to add more features and polish the UI of the Padawan Wallet.
I have thought of some features:

- Gamification features: To increase user engagement and motivation, we can introduce gamification features such as achievements, badges, or leaderboards. This will incentivize users to continue learning and using the app.

- Community features: Introduce community features such as discussion forums, Q&A sections, or peer-to-peer mentorship programs, allowing users to connect with other learners and experts in the field.

- Multi-currency support: Besides Bitcoin, the app can support other cryptocurrencies, allowing users to learn about different wallets and their unique features.

It would give me immense pleasure to be a part of the Padawan community and help people learn about cryptocurrency and blockchain technologies.

Benefits to Community

Adding more features and improving the app's overall polish, and the project can help individuals better understand how Bitcoin works and how to use it safely and effectively. This can ultimately lead to more widespread adoption of Bitcoin and other cryptocurrencies, which can have significant economic benefits for communities.

Biographical Information

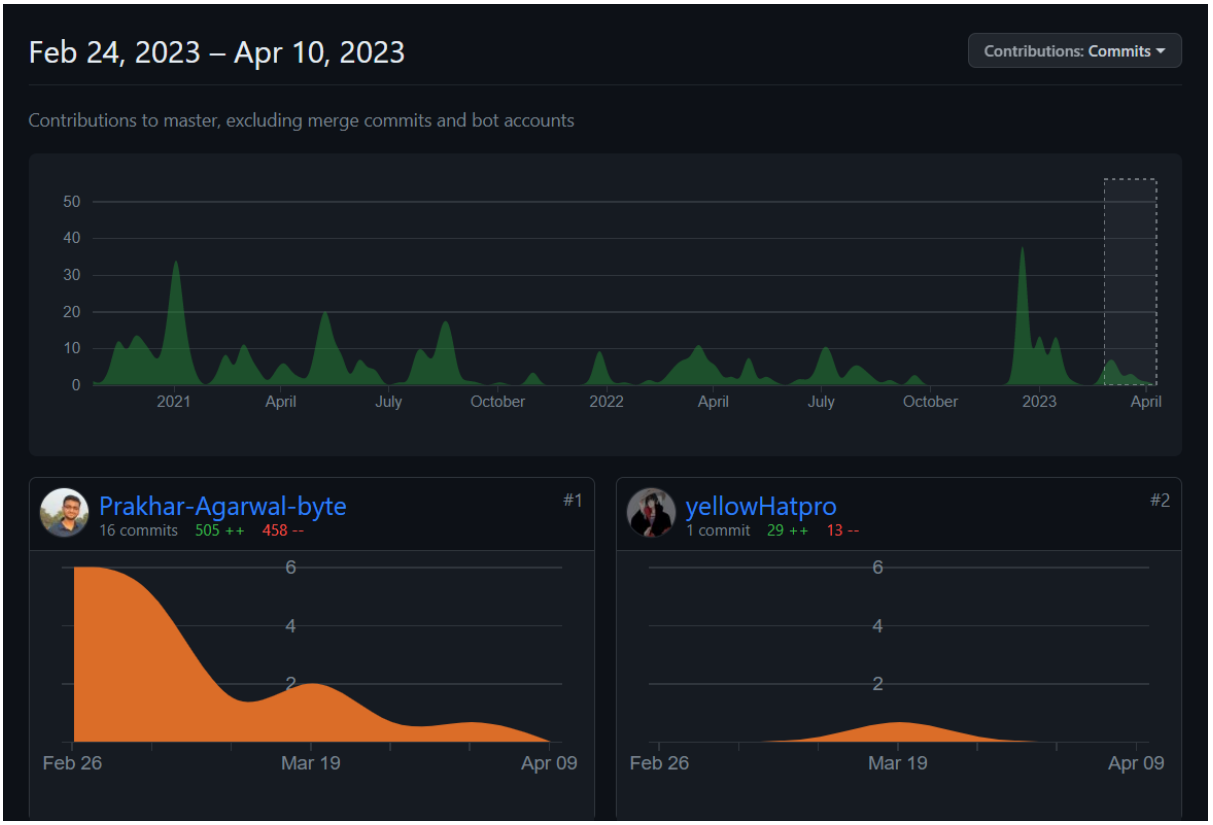
I am **Prakhar Agarwal**, a 3rd-year student at the Indian Institute of Technology (BHU) Varanasi. I am an Android Developer proficient in **Kotlin, Java, Jetpack Compose, MVVM architecture, and Git**. I have been tinkering with code from a young age and am obsessed with technology. I love exploring new tech and have worked with various technologies and languages. I love contributing to the open-source community and building something that can benefit the community.

Padawan Wallet

I have been contributing to the Padawan project since February 2023 and have fixed several bugs and introduced new features to the app. A list of all my contributions till now is given below:

PR	Issue	Description	Status
#263	#257	Display amount in accepted standard	Merged
#254	#252	Show sync status	Merged
#249	#247	Fix sliding transition	Merged
#300	#296	Increase finish button padding	Merged
#274	#272	Sync button touches the card's bottom edge	Merged
#264	#258	Fix navigation on the Menu screen	Merged
#260	#256	Add snackbar to show chapters reset was successful	Merged
#242	#241, #201	Show copied to clipboard snackbar	Merged
#240	#239	Made privacy policy link clickable	Merged
#251	#250	Fixed broken links	Merged
	#297	Highlight the Bottom Navigation Tab	WIP

Since I started contributing, I have been the top contributor with the most commits, PRs, Issues raised, and lines of code changed.



I also did code reviews of other contributors on #294 and #295. Apart from this, I am actively in contact with the mentor **@thunderbiscuit#7768** over Discord and participate in discussions on GitHub.

Google Summer of Code 2021 with Purr Data

I was selected for Google Summer of Code 2021 with Purr Data organization. During the project, I improved the overall usability of the Purr Data Web Application to make it more intuitive, improved its stability by fixing many bugs, and improved the UI to fit the web app better. Technologies I used in this project - **Javascript, HTML, CSS, NW.js, C++, Bootstrap**.

You can find details of my work on the GSoC project at
https://github.com/Prakhar-Agarwal-byte/purr-data

Link to the Repo: **Purr Data**
My Pull Requests: **PR**
Issues Raised: **Issues**

Contributor at Amahi

I have contributed to the Amahi Android app codebase and received a **recommendation letter from its co-founder, Carlos Puchol**. I worked on improving the UI/UX of the Android app, fixed bugs, and enhanced many features in it. Technologies I used in this project - **Java, Kotlin, Android**

Link to the repo: **Amahi**
My Pull Requests: **PR**
Issues Raised: **Issues**

Contributions to other open-source projects

I've contributed to the **Meshery** and **Layer-5**, a cloud-native management plane offering lifecycle, configuration, and performance management of service meshes. Here I **fixed issues, reviewed PRs,** and learned to **work collaboratively**.

Link to the Repo: **Meshery**
My Pull Requests: **PR**
Issues Raised: **Issues**

Link to the Repo: **Layer-5**

My Pull Requests: **PR**

Internships at startups

I have **done many internships at startups as an Android developer**, where I got to work in a team and enhance my technical expertise with the help of my mentors. Technologies I used: **Android, MVVM, Retrofit, Jetpack Compose, Glide, RoomDB, Kotlin, Java.**

- **MentorPlus - Android Developer Intern**
  • Developed the Android app of MentorPlus to host interactive mentor-mentee learning sessions via WebRTC
  • Implemented **authentication**, referral-based user onboarding, and refactored codebase to MVVM architecture. Integrated Razorpay payment gateway and Firebase analytics
  • Utilized **Agora SDK** to develop screen share functionality which improved user engagement by **31%**
  • Technologies Used: **Kotlin, MVVM, Firebase, AgoraSDK, Glide, Retrofit**

- **Entwik Studios - Android Developer Intern**
  • Worked on the app which creates lobbies/matches for players with similar interests
  • Integrated Paytm payment service for seamless online payments. This increased the paid user count by **13%**
  • Added OTP-based authentication using SMS Retriever API and auto app updates using Google Play API
  • Technologies Used: **Kotlin, PaytmSDK, OTP Auth, Retrofit, Google Play**

Ecommerce Warranty using NFT

I and my friend developed a warranty system for e-commerce using NFTs. This would allow the customer to get the purchasing history, warranty period, track repairs and replacements to the original item. Customers can use digital NFT to verify the authenticity of their products, prove their ownership, and transfer ownership upon resale. Technologies Used: **Solidity, Polygon Testnet, Blockchain, NFT, Javascript, and React**

Link to the repo:
https://github.com/Hitansh-Shah/ecommerce-warranty/

I have all the relevant skills and experience required for this project, and I am sure that using my knowledge and with the help of mentors we can make this project a grand success.

References

- https://guide.summerofbitcoin.org/
- https://github.com/thunderbiscuit/padawan-wallet/issues/238
- https://developer.android.com/training/testing/fundamentals
- https://github.com/thunderbiscuit/devkit-wallet
- https://bitcoindevkit.org/
- https://github.com/thunderbiscuit/tatooine
- https://discord.com/channels/857861349860966401/1073876515453927434