Summer of Bitcoin 2024

Proposal

Multi-Account Feature in Blink Wallet app

Prakhar Agarwal

Table of Contents

Personal Information

Name:	Prakhar Agarwal
Github:	Prakhar-Agarwal-byte
Discord:	prakhara1
Email:	prakharagarwal3031@gmail.com
College:	Indian Institute of Technology (BHU), Varanasi
Degree:	Bachelor of Technology
Linkedin:	prakhar-agarwal-byte
Country:	India
Timezone:	IST (UTC+5:30)
Phone:	+919997793031

Project Overview

Galoy

Galoy is a cloud-based platform that allows users to integrate Bitcoin into their organizations. It offers services such as money transfers, QR-based payments, and lightning payments.

Blink Wallet

Blink Wallet is the official mobile app that uses the Galoy backend for trasacting Bitcoins.

Multi-Account feature in Blink Wallet

Users should be able to log in to the Blink app with multiple user accounts and be able to manage their accounts.

Expected Outcomes

- Multi-account log-in feature
- Users should be able to:
  - log in to multiple accounts
  - log out of accounts selectively
  - switch between accounts
- Notifications must differentiate between accounts

Implementation Details

The Blink app at present allows for only one account per app, although a user might possess multiple accounts. Allowing the user to oversee all their accounts within a single app would enhance convenience. Users can effortlessly switch between their accounts whenever necessary.

First we need to study how authentication and token management works in Galoy stack. Here's what I found:

Present Implementation

A user can log in using either email or mobile; the authentication flow is similar in both; here, I am demonstrating the **email login flow**. The user enters his email id and receives a code. We check if the code is valid in **email_login_validate.tsx**,

galoy-mobile - email-login-validate.tsx

```
42 const url = `${authUrl}/auth/email/login`
43
44 const res2 = await axios({
45   url,
46   method: "POST",
47   data: {
48     code,
49     emailLoginId,
50   },
51 })
```

Implementation Details

```
res2.data.result = {
  authToken,
  totpRequired,
  id
}
```

The result is an object that has **authToken, totpRequired, and id**

If an auth token is present, and totp is not required, the token is saved on device.

galoy-mobile - email-login-validate.tsx

```
58 if (authToken) {
59   if (totpRequired) {
60     navigation.navigate("totpLoginValidate", {
61       authToken,
62     })
63   } else {
64     analytics().logLogin({ method: "email" })
65     saveToken(authToken)
66     navigation.replace("Primary")
67   }
68 } else {
69   throw new Error(LL.common.errorAuthToken())
70 }
```

saveToken calls **updateState** function to set the **authToken** in **PersistantState**, which in turn updates the **appConfig** as it is present as a dependency

galoy-mobile - use-app-config.ts

```
9  const appConfig = useMemo(
10   () => ({
11     token: persistentState.galoyAuthToken,
12     galoyInstance: resolveGaloyInstanceOrDefault(persistentState.galoyInstance),
13   }),
14   [persistentState.galoyAuthToken, persistentState.galoyInstance],
15 )
```

Implementation Details

galoy-mobile - client.tsx

```
69 const GaloyClient: React.FC<PropsWithChildren> = ({ children }) => {
70   const { appConfig } = useAppConfig()
```

GaloyClient has a **useEffect** with appConfig.token dependecy. When the token is updated it creates a new client with proper **Authorization headers**.

galoy-mobile - client.tsx

```
110 const authHeaders = token ? { Authorization: getAuthorizationHeader(token) } : {}
```

Throughout the app this client is used using the **ApolloProvider** to make graphQL API calls, which contain user's auth token.

galoy-mobile - client.tsx

```
322 <ApolloProvider client={apolloClient.client}>
323   <IsAuthedContextProvider value={apolloClient.isAuthed}>
```

Feature Approach

Support multiple account

When API call is made at the URL **authUrl** it returns the **authToken** as well the **kratosUserId**.

galoy-mobile - email-login-validate.tsx

```
42 const url = `${authUrl}/auth/email/login`
```

galoy - login.ts

```
187 return {
188   authToken: res.authToken,
189   totpRequired,
190   id: res.kratosUserId
191 }
```

We now **migrate** the PersistantState schema to have a **list of account** objects that will have the number or email, kratosUserId and token for that account.

galoy-mobile - state-migrations.ts

```
27 type PersistentState_7 = {
28   schemaVersion: 7
29   galoyInstance: GaloyInstanceInput
30   galoyAuthToken: string
31   galoyAccounts: {
32     id: string
33     email: string
34     token: number
35   }[]
36 }
```

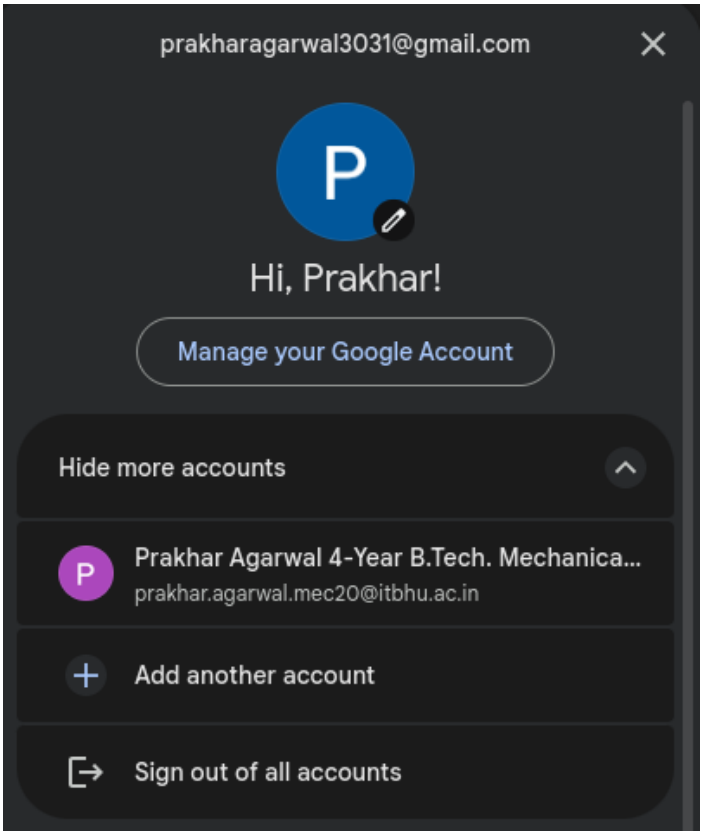
Feature Approach

When the user logs in, we will store the **email, ID, and token** in the **galoyAccounts** list. The galoyAuthToken will continue to always store the current user's token. The graphQL client will observe the change in token and configure the client with the updated token.

galoy-mobile - use-app-config.ts

```
const saveAccount = useCallback(
  ({email, id, token}: { email: string; id: string; token: string }) => {
    updateState((state) => {
      if (state) {
        return {
          ...state,
          galoyAccounts: [
            ...state.galoyAccounts,
            {
              emailId: email,
              id,
              token,
            },
          ],
        }
      }
      return undefined
    })
  },
    [updateState]
)
```

Feature Approach



In the settings screen, inside Account settings, we will create a select box to **switch** between the saved accounts. Also put an option to log into a new account.

We can take inspiration from how Google lets us switch between accounts.

When the user selects an account, we search the PersistentState for the account, take the token from there, and update the **galoyAuthToken**. The graphQL client will observe the change in token and configure the client with the updated token. From here, everything will work the same.

galoy-mobile - banner.tsx

```
const { saveToken } = useAppConfig()
const { persistentState} = usePersistentStateContext()

const switchAccount = (emailId: string) => {
  const filteredAccounts = persistentState.galoyAccounts.filter(account => account.emailId === emailId)
  if (filteredAccounts.length > 0) {
    const selectedAccount = filteredAccounts[0]
    saveToken(selectedAccount.token)
  }
}
```

Feature Approach

When a user logs out, we reset the galoyAuthToken, instance, and schemaVersion, but we **keep the accounts list intact**. This ensures that the current user get logged out but can then log in to a different saved account.

galoy-mobile - index.tsx

```
55 const resetState = React.useCallback(() => {
56   setPersistentState({
57     ...persistentState,
58     schemaVersion: 6,
59     galoyInstance: { id: "Main" },
60     galoyAuthToken: "",
61   })
62 }, [])
```

Handling notifications for different accounts

A user must only see the messages meant for the currently logged-in account. To handle this, we will embed the token in the notification message and filter out the notifications we display on the client side. If the notification's token matches the current user's authorization token, then only we display the notification; otherwise, we just ignore it.

galoy-mobile - push-notification.tsx

```
77 const unsubscribeInApp = messaging().onMessage(async (remoteMessage) => {
78   if (remoteMessage.data?.token == persistentState.galoyAuthToken) {
79     notifyCard({
80       text: remoteMessage.notification?.body ?? "",
81       title: remoteMessage.notification?.title ?? "",
82       action: async () => {
83         showNotification(remoteMessage)
84       },
85       icon: "bell",
86     })
87   }
88 })
```

Timeline

Time Period	Tasks
May 8 - May 15	Discuss and finalize project implementation plan, milestones, timeline with my mentor
May 15 - May 30	• Collect account information that needs to be saved • Create data structure for storing account information in device storage
May 30 - June 15	• Configure PersistantState schema • Write migrations for the new schema
June 15 - June 30	• Work on handling tokens when switching accounts • Write tests to ensure account changes correctly • Implement log out feature
June 30 - July 15	• Design and develop UI components for Account switching • Complete navigation and animations while changing accounts
July 15 - July 30	• Write E2E tests • Complete documentation for the feature • Beta Testing
July 30 - Aug 15	Deploy in Production

About Me

I am **Prakhar Agarwal**, a final-year student at the Indian Institute of Technology (BHU) Varanasi. I am a Full Stack Developer proficient in **React Native, Android, and Web Development using JS, TS, Golang, Java, and Kotlin**. 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.

Galoy

I have been involved with Galoy since **March**. I have already set up dev env and I am familiar with the codebase. Also I have created a couple of PRs fixing some issues.

PR	Issue	Desciption
#3143	#3055	fix: confirm payment slider bug in rtl layouts
#3145	#3128	feat: Add link opening functionality in ScanningQRCodeScreen
#4271	#4252	Auto-set display currency for level 1 users based on phone number prefix

Experience

Summer of Bitcoin 2023 - Padawan Wallet

- Developed advanced features for the Padawan Bitcoin Wallet using the Bitcoin Development Kit.
- Integrated OP_RETURN opcode, BIP32 address paths, BIP84 to generate wallet addresses, BIP21 for payment links.
- Migrated to Kotlin Flows from LiveData, improved UI/UX/animations, and made numerous fixes for small screens. Link Code

Google Summer of Code 2023 - Purr Data

- Implemented an Autosave feature in Purr Data which creates configurable file backups to prevent against data loss.
- Maintained a JSON-based configuration file to store key-value pairs that map original to autosaved file paths.
- Optimized the storage utilization by 57% by tracking changes in content and keeping only the unrecovered files. Link Code

Experience

LFX Mentorship [CNCF - Konveyor: Move2Kube]

- Analyzed WASM for in-browser execution of Move2Kube (IaC automation tool), analyzing feasibility and challenges.
- Investigated workarounds for sys/unix and syscall package usage in Move2Kube to facilitate WASM compatibility.
- Proposed solutions for network limitations and file system access in a WASI environment for the Move2Kube CLI. Link Code

Google Summer of Code 2021 - Purr Data

- Designed frontend of the app while incorporating best UI/UX design principles, making it more intuitive to use
- Made shortcuts work depending on the device platform (MacOS, Windows, Linux)
- Improved the file manager so files/folders can be added/renamed/deleted and initial loading time of app by 12% Link Code

Availability

Availability

I have my official college **summer break** from the **10th of May to the 7th of August**, and I have no other commitments during this period, so I will be able to devote a minimum of **35-40 hours per week**. I will also be accessible full-time during weekends after this and keep the community updated about my progress.

Equipment

I have a **Ubuntu dual-boot top-end laptop** with high-speed internet connectivity. I have a good pair of headsets and a webcam, so there is absolutely no issue for me in attending video meetings as well.

Language/Communication

I have a full working proficiency in **English** and am native Hindi speaker

Timezone/Remote Collaboration

My timezone is IST (UTC+5:30). Since I have my holidays, I do not have any time restrictions and I am **flexible to work** in any time slot as required.